

CHAPTER 3

*i*AS Configuration
and Tuning

IAS is a highly configurable and tunable product that can meet the needs of a wide range and type of Web sites. Configuring *iAS* requires an understanding of the configuration of the Apache Web server and its various modules, such as `mod_jserv` and `mod_plsql`. To take advantage of features in *iAS* while maximizing performance, the *iAS* administrator should be well versed in all the features, and their associated costs and benefits.

In this chapter, the following topics are discussed:

- Quick Tuning and Configuration Answers
- Getting Started
- Understanding Ports
- Starting, Stopping, and Restarting *iAS* (Apache)
- Configuring *iAS* (Apache)
- GUI *iAS* Configuration Editing
- Managing and Configuring Modules
- Configuring *iAS* Security
- Monitoring Your *iAS* Web Server
- Tuning *iAS*
- Tuning Application Modules
- Hardware and Operating System Tuning

Quick Tuning and Configuration Answers

If you prefer to read the quick answer rather than the entire chapter, this section is for you. In this section, I summarize the major points of configuration for optimum performance and security. Because summarizing an entire chapter into one section is difficult, I highly recommend you read the entire chapter. If you don't have time, the following is the short version:

- Hardware
 - Size your hardware properly
 - At a minimum, distribute *iAS*, database, and FTP on to individual servers

- Hardware components should operate at no more than 75 percent to 80 percent of capacity
- Use `directio` for maximum disk performance
- Distribute the load
- The more RAM, the more *iAS* and module instances can be invoked
- Production servers should have at least 1GB of RAM
- Operating System
 - Use the logs and Web utilities to monitor *iAS* use
 - Use `top`, `vmstat`, `netstat`, `sar`, and `ps` to manage memory use, paging, swapping, and TCP/IP parameters on UNIX
 - Use Task Manager and Performance Monitor to manage memory use, paging, and swapping on NT/2000
 - Set the operating system swap file size to three times the physical memory
 - Set TCP/IP parameters per the hardware vendor's specification for a Web server
 - Obtain the latest operating system patches, especially for TCP/IP
- *iAS*
 - Tune *iAS* (Apache) directives and parameters
 - Keep logs small (1–5MB)
 - Log only warnings and errors (LogLevel warn)
 - Monitor logs for errors
 - Become familiar with the default configuration files. They provide good configuration information and settings
 - Don't use SSL unless it's a requirement
 - Turn off (comment out) modules you aren't using
 - Avoid using the `rewrite`, `usertrack`, `session`, and `extended status` modules
 - Turn off `.htaccess` capability
- Modules
 - PL/SQL

- Make use of OWA_CACHE for better PL/SQL performance
- Set session state to true if you have few users executing a series of PL/SQL procedures
- Use connection pooling
- Pin key packages in memory
- Perl
 - Turn on caching
- Java
 - Make use of JDBC connection pooling
 - Pin key classes in memory
 - Turn auto reload off
 - Use the single-threaded model
 - Load balance JServ
 - Be sure debug mode is off
- Oracle Database and SQL
 - Tune your database
 - Use Oracle9i rather than Oracle8i
 - Tune your PL/SQL and SQL code

Getting Started

iAS is based on Apache. Entire books are written about configuring Apache. For more detailed information, I highly recommend *Professional Apache*, by Peter Wainwright (Wrox Press, 1999). This chapter's focus is specifically for *iAS* and the configuration components of primary interest to you.

Prior versions of the Oracle Application Server (OAS) included a graphical browser-based configuration tool. Apache provides no such tool and neither does *iAS*. The administrator is expected to maintain the configuration files using a text editor (such as *vi*, Notepad, or TextPad). Future versions of Oracle Enterprise Manager (OEM), included with 9*iAS*, are expected to include the capability to configure Apache. Until then, you need to understand these configuration files. Understanding how Apache works is extremely important.

Where iAS Gets Its Configuration Information

Prior to version 1.3.4, Apache used the following three configuration files located within the conf subdirectory of the server's root directory (usually the \$ORACLE_HOME/Apache/Apache/conf directory):

- Master configuration file, typically named httpd.conf, which contains the basic server configuration, including the locations of the other two configuration files.
- Resource configuration file, specified by the ResourceConfig parameter in the master configuration file, and typically named srm.conf, which contains the resource directives.
- Access configuration file, specified by the AccessConfig parameter within the master configuration file, and typically named access.conf, which contains the directory access control directives.

All three of these files are still included in the Apache installation, but the resource and access configuration files are now empty and their directives are placed in the master configuration file. Because Oracle 9iAS 1.0.2.2 uses Apache 1.3.19, the resource and access configuration files aren't used.

Understanding Directives

The configuration files contain directives to tell iAS how to operate. A *directive* or command is an instruction that tells iAS to perform a specific task. iAS responds to a specific set of commands. Generally, the directive name is followed by a series of one or more arguments. Directives typically start with an uppercase letter and each word within the directive is also capitalized. Directives that contain directives within them, *subdirectives*, are typically surrounded by a less than (<) and a greater than (>) sign, just like XML and HTML. For example, the directive for virtual hosts is <VirtualHost>. Here are some directives pulled from the httpd.conf file:

```
ServerType standalone
Port 80
ServerName tuscdbd
DocumentRoot "C:\ORACLE\iSuites\Apache\Apache\htdocs"
<Directory />
    Options FollowSymLinks
    AllowOverride None
</Directory>
```

If you're curious about the possible directives, use `apache -L` as defined in the following "Starting, Stopping, and Restarting iAS (Apache)" section.

Oracle Changes to the Master Configuration File

The Apache master configuration file created during the Oracle 9iAS installation has changed from the standard Apache master configuration file. Some of the changes made to the configuration file include the following:

- Minor filename changes, such as for the values of the LockFile and ScoreBoardFile parameters.
- SSL support by adding a Listen 443 directive and including the mod_ssl.c module, specifying the parameters for the global context of the mod_ssl.c module, and specifying the parameters for the SSL virtual host context directive <VirtualHost _default_:443>.
- An alias named /jservdocs/ to point to the physical directory where the Apache JServ documentation is installed. Apache JServ replaces Oracle JWeb used in prior versions of the Oracle Application Server.
- A Dynamic Monitoring System (DMS) directive introduced in 9iAS 1.0.2.1 to allow a browser to access performance and resource metrics out of the shared memory scorecard. If this section is removed, DMS metrics are still collected, but cannot be accessed by a browser.
- An alias named /perl/ to point to the physical directory where Perl scripts are stored, and the appropriate directives for processing these scripts in the persistent Perl run-time environment embedded in the HTTP server.
- Directives to include two of the configuration files that contain the Oracle-specific directives.

iAS Configuration File Structure

iAS uses eight configuration files containing the required directives. Two of the Oracle configuration files—`jserv.conf` and `oracle_apache.conf`—are included at the end of the master configuration file. The other Oracle configuration files are included using directives within the `oracle_apache.conf` file. In other words, they're nested. See Figure 3-1 for the hierarchy of all the standard Apache and Oracle configuration files.

The following is a summary of the information contained in each of the Oracle configuration files:

- **jserv.conf** Apache JServ configuration that loads the JServ module and specifies the appropriate module parameters.
- **oracle_apache.conf** Includes the remaining six configuration files and defines a virtual directory for Discoverer files.

- **mod_ose.conf** Oracle Servlet Engine (OSE) configuration that loads the OSE module and specifies the appropriate module parameters. The OSE module services requests to stateful Java and PL/SQL servlets.
- **plsql.conf** PL/SQL configuration that loads the PL/SQL module and specifies the appropriate module parameters. This file also specifies the aliases for the /help/ and /images/ virtual directories used by Oracle Portal, plus directives specifying the handling of the /images/ and /docs/ subdirectories of the Portal installation. Note, the /images/ alias is also defined in another Oracle configuration file—6iserver.conf.
- **ojsp.conf** JavaServer Pages (JSP) configuration that defines the /jspdocs/ alias and associates requests containing a .jsp or .sqljsp extension to the JServ module.
- **xml.conf** Extensible Markup Language (XML) configuration that defines the /xsql/ alias and associates requests containing an .xsql extension to the XSQL servlet.
- **6iserver.conf** Forms and Reports configuration that defines aliases, environment variables, and MIME types used by the Forms and Reports servers. As noted previously, the /images/ alias in this file is ignored because it's previously defined in plsql.conf.

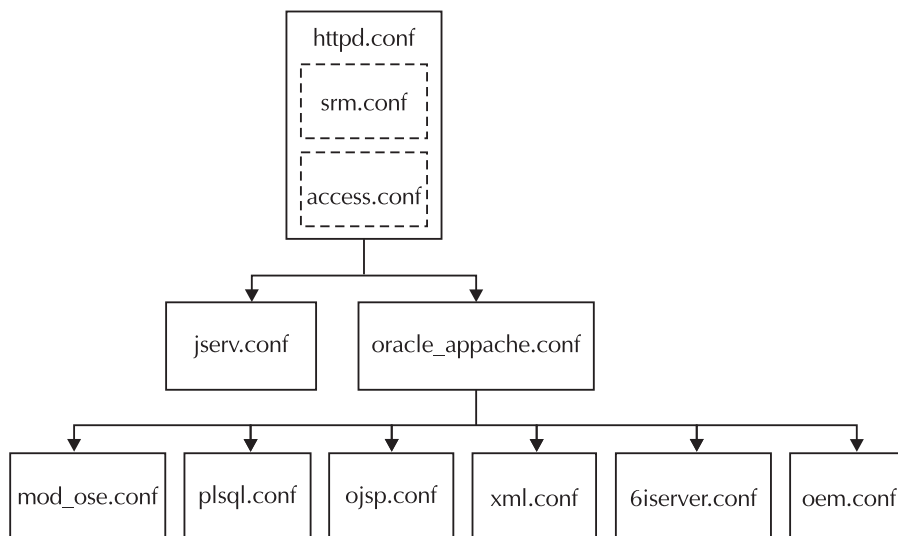


FIGURE 3-1. Standard Apache and Oracle configuration files

- **oem.conf** Oracle Enterprise Manager (OEM) configuration that defines the listen port (default is 3339), virtual host `_default_`:port, and associated virtual host directives to use the browser-based interface of OEM.

More details about the configuration of *iAS* will be discussed shortly, but first, let's discuss TCP/IP ports and the *iAS* (Apache) control program.

Understanding Ports

Ports are virtual divisions within an IP address on a machine. Even though a listener can operate on any port (providing you have the privilege and the port doesn't conflict with another service), in general, ports with a number less than 1025 are used as system-level ports—typically assigned by the Transmission Control Protocol/Internet Protocol (TCP/IP) commission that sets standards for these port numbers. Port 80 is the industry standard TCP/IP port for HTTP communication.

To see which ports *iAS* has used, look at the file `$ORACLE_HOME/portlist.txt`. This file shows a list of the ports, but it doesn't show what they're used for and this isn't an inclusive list. For example, here's the `portlist.txt` file from my machine:

```
80
443
8007
7777
4443
7778
8008
```

The previous list doesn't include many ports, such as the Web Cache ports (for example, 4000, 4001, 4002). To see which ports the system is using, use the following line command:

```
netstat -a
```

The `netstat` command displays protocol statistics and current TCP/IP network connections. The `-a` parameter displays all connections and listening ports. Executing this command shows you an extensive listing of every port in use on your machine. The following is only a partial listing for my laptop:

```
Active Connections
```

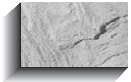
Proto	Local Address	Foreign Address	State
TCP	TUSCBDB:2989	TUSCBDB:0	LISTENING
TCP	TUSCBDB:8080	TUSCBDB:0	LISTENING
TCP	TUSCBDB:9001	TUSCBDB:0	LISTENING
TCP	TUSCBDB:15000	TUSCBDB:0	LISTENING
TCP	TUSCBDB:990	TUSCBDB:0	LISTENING


```

TCP      TUSCBDB:990          TUSCBDB:2985          ESTABLISHED
TCP      TUSCBDB:4900        exchange.tusc.com:smtp TIME_WAIT
TCP      TUSCBDB:8080        TUSCBDB:4870         TIME_WAIT
TCP      TUSCBDB:8080        TUSCBDB:4889         TIME_WAIT
TCP      TUSCBDB:8080        TUSCBDB:4894         TIME_WAIT
TCP      TUSCBDB:8080        TUSCBDB:4897         TIME_WAIT

```

...

**TIP**

If you plan to put a listener on a port other than 80 or 443, you should use a number greater than 1024, so you don't conflict with industry standard port numbers. Most proxy servers limit users to ports 80 and 443 for HTTP and HTTPS. If you plan to deploy an application on the Internet, be aware that choosing a port outside these numbers can cause proxy issues for your users. However, because port 8080 was the de facto standard HTTP port, many proxy servers also allow access to port 8080.

The following table outlines the standard port assignments and uses:

Port Number	Standard Assignments
80	The default HTTP Listener port
443	The default HTTPS Listener (SSL—Secure Sockets Layer) port
21	The FTP (File Transfer Protocol) port
23	The telnet port
79	Finger daemon
110	POP-3
Ports 1–1024	Reserved for system level duties

Starting, Stopping and Restarting iAS (Apache)

The iAS control program is respectively named `apache` on NT/2000 and `apachectl` on UNIX. To start iAS on NT/2000 at the command line, check to see that the `apache` executable is in the system PATH environment variable. To check this, find the `apache` executable, typically found under the `$ORACLE_HOME/Apache/Apache` directory, and then check the path by selecting the system icon in the Control Panel and selecting the Environment tab (NT) or Advanced tab (2000). In most of my

installations, the PATH variable wasn't set on NT/2000 for the \$ORACLE_HOME/Apache/Apache directory. Add the directory for the apache executable in the PATH environment variable if it isn't set.

To start *iAS* from the command line, enter the following:

```
apache -k start
```

To stop *iAS* from the command line, enter the following:

```
apache -k shutdown
```

You can also start and stop *iAS* on NT/2000 using the Oracle HTTP Server shortcuts under the Start Button | Programs | Oracle. Additionally, the Services icon in the Control Panel has an entry for Oracle HTTP Server, which enables you to start and stop *iAS*.

To start *iAS* on UNIX, check to see if the path contains the directory holding the httpd executable, using the following line command:

```
echo $PATH
```

To start *iAS* on UNIX from the command line, enter the following:

```
apachectl start
or
httpd
```

To have *iAS* start automatically when the server is rebooted, edit /etc/rc.d/rc.local and add \$ORACLE_HOME/Apache/Apache/bin/httpd

To shut down *iAS*, enter

```
apachectl shutdown
or
kill -9 `cat $ORACLE_HOME/Apache/Apache/logs/httpd.pid`
```

Command line syntax for apache and apachectl is

```
Usage: APACHE [-D name] [-d directory] [-f file] [-n service]
             [-C "directive"] [-c "directive"] [-k signal]
             [-v] [-V] [-h] [-l] [-L] [-S] [-t] [-T]
```

Command line options for UNIX and NT/2000:

- **-D name** Define a name for use in <IfDefine name> directives
- **-d directory** Specify an alternate initial ServerRoot
- **-f file** Specify an alternate ServerConfigFile (good for testing configuration changes or having multiple installations of *iAS* server)

- **-C “directive”** Process directive before reading configuration files
- **-c “directive”** Process directive after reading configuration files
- **-v** Show version number
- **-V** Show compile settings
- **-h** List available command line options (this page)
- **-l** List compiled-in modules
- **-L** List available configuration directives
- **-S** Show parsed settings (currently only vhost settings)
- **-t** Run syntax check for configuration files (with docroot check)
- **-T** Run syntax check for configuration files (without docroot check)
- **-n name** Name the *iAS* service for the following **-k** options:
 - **-k stop | shutdown** Tell running *iAS* to shut down
 - **-k restart** Tell running *iAS* to do a graceful restart
 - **-k start** Tell *iAS* to start
 - **-k install | -i:** Install an *iAS* service
 - **-k config** Reconfigure an installed *iAS* service
 - **-k uninstall | -u** Uninstall an *iAS* service

Configuring *iAS* (Apache)

To configure *iAS* and its many modules manually, you need to be familiar with the *iAS* directory structure and the location of the configuration files. To locate the configuration files, see the following directories, which are in `$ORACLE_HOME/` `Apache/` `Apache`:

- **bin** Location of *iAS* executables
- **cgi-bin** CGI programs
- **conf** *iAS*/Apache
- **htdocs** Location of Web site files, such as HTML files
- **logs** Log files

Oracle Configuration Files

In addition to the previously defined configuration files in the “*iAS Configuration File Structure*” section, additional *iAS* configuration files include:

- **mime.types** Multipurpose Internet Mail Extensions (MIME) mapping between MIME types and file extensions
- **wdbsrv.app** Configuration information for Database Access Descriptors (the PL/SQL module)
- **formsweb.cfg** Parameters used by forms CGI programs
- **jserv.properties** Configuration for JServ engine
- **zone.properties** Settings for servlet zone

As previously mentioned, the most important and driving configuration file in *iAS* is the `httpd.conf` file, located in the `$ORACLE_HOME/Apache/Apache/conf` directory. This file contains directives for the Web sites and the behavior of *iAS*. The file is read at the startup of *iAS*, so changes to this file don’t take effect until *iAS* is restarted. The directives in *iAS* follow a bottom up inheritance model, in which case settings are inherited or overridden from bottom to top for the file. If two contradicting directives exist, the last directive takes precedence over the first directive. Directives also scoped into three tiers: server, container, and per-directory.

Server Directives

The server directives are global to *iAS*. For example, the directives specifying `ServerRoot` and `PidFile` are global directives, as follows:

```
ServerRoot "D:\Oracle\Ora81\Apache\Apache"
```

and

```
PidFile logs\httpd.pid
```

Container-Level Directives

The *container level* is limited in scope for directives and overrides server-level directives. They provide a way of limiting scope through a specific directory, URL, or HTTP request. In this example, I tell *iAS* to allow access to server status information from the localhost only, so if users try to find out information about the host server and their request isn’t from the localhost, they’ll be denied access.

```

<Location /server-status>
    SetHandler server-status
    Order deny,allow
    Deny from all
    Allow from localhost
</Location>

```

Per-Directory and File Directives

The *directory level* (per-directory) directive has directory scope, meaning it applies to a specified physical directory and overrides container and server directives. Directory level is a commonly used directive to notify iAS about what directories are accessible to the Web site visitors. In the following example, I enable the htdocs directory to be accessed by all, and to have symbolic links and server-side includes.

```

<Directory "c:\website/example1/htdocs/">
    Options Indexes FollowSymLinks +Includes
    AllowOverride None
    Order allow, deny
    Allow from all
</Directory>

```

Directory directives levels can use wildcards and search patterns. In the following example, I tell iAS to grant visibility to the docs directory and any of its subdirectories.

```

<Directory C:\website\example1\docs\*>
    Order allow, deny
    Allow from all
</Directory>

```

File directives apply to specified files. In the following example, I tell iAS not to allow files with the extension .ht to be viewed because .ht files contain security and directory information, and I want to limit access to that information. File directives can also use wildcards and search patterns.

```

<Files ~ "\.ht">
    Order allow, deny
    Deny from all
</Files>

```

Some directives, like Limit and FileMatch, can be nested within a directory.

Directive Merging

When iAS starts up, it parses the httpd.conf file and performs directory merging. iAS merges the following: Directory and .htaccess, Location and LocationMatch,

Files and FileMatch. .htaccess files are unique because they are parsed and followed for every request.

Directives can be placed inside other directives. The following directives can be placed inside directory and file directives:

- **ExecCGI** Allow execution of CGI programs
- **Includes** Allow server-side includes (SSI)
- **Indexes** Allow directory indexing
- **FollowSymLinks** Allow files accessed using symbolic links
- **SymlinksIfOwnerMatch** Allow symbolic links only if linked file is owned by the owner of the link
- **All** Enable all options except multiview
- **Multiview** Content negotiation
- **None** Disable all options

Setting Your Global Directives

ServerType can be set to either inetd or standalone. inetd mode is supported only on UNIX platforms and isn't a common setting. The Default mode is standalone, which allows Apache to handle its own network connections. For example:

```
ServerType standalone
```

ServerName is the name of the server responding to the HTTP requests. It can be a hostname or IP address. For example:

```
ServerName localhost  
ServerName 127.0.0.1
```

ServerRoot is the default location where the configuration, error, and log files are kept. For example:

```
ServerRoot "D:\Oracle\Ora81\Apache\Apache"
```

PidFile is the file in which the server records the process identification number for the *iAS* process when it starts. This process ID can be used to stop or restart the server. For example:

```
PidFile logs\httpd.pid
```

Timeout is the number of seconds before the server time out. For example, to timeout in 300 seconds (5 minutes):

```
Timeout 300
```

KeepAlive tells Apache whether to allow persistent connections. For example, to keep connections alive:

```
KeepAlive On
```

BindAddress is the directive used to tell the server which IP address to listen to. BindAddress can contain "*", an IP address, or a fully qualified Internet domain name. For example, to bind all addresses:

```
BindAddress *
```

Port is the default port for HTTP servers to operate. Port 80 is the default port. For example:

```
Port 80
```

Listen is a directive that can be used to replace Port and BindAddress directives. Listen can be specified multiple times for a list of ports. Listen is defaulted for port 80 and uses 443 for SSL. For example:

```
Listen 443
```

ServerAdmin is the e-mail address to which problems with the server should be sent. For example:

```
ServerAdmin Bradley_d_brown@tusc.com
```

DocumentRoot is the default location where Web pages reside. By default, iAS looks for the htdocs directory under ServerRoot. DocumentRoot sets the virtual root for the initial HTML document directory. For example:

```
DocumentRoot "D:\Oracle\Ora81\Apache\Apache\htdocs"
```

ErrorLog is the location of the error log file. For example:

```
ErrorLog logs\error_log
```

LogLevel is used to set the logging level condition. Error, debug, and warn are some common settings. As discussed in the following, for better performance, the preferred setting is error, but the default setting is warn. For example:

```
LogLevel error
```

The Include directive provides the capability to modularize configuration files, and allows the administrator the flexibility of containing module specific directives

in separate files. Examples include `jserv.conf`, which contains directives for the Java Servlet Engine, and `oracle_apache.conf`, which has directives for additional includes, such as the `plsql.conf` and the `ojsp.conf`. Includes can be referenced using the full path or a relative path. Include files can be used for *iAS* installations supporting multiple Web sites because the file for all the Web site configurations may be dynamically generated.

iAS uses the following include files:

```
include "D:\oracle\ora81\Apache\Jserv\conf\jserv.conf"
include conf\oracle_apache.conf
include "D:\Oracle\Ora81\Apache\jsp\conf\ojsp.conf"
include "D:\Oracle\Ora81\Apache\modplsql\cfg\plsql.conf"
```

If you want to put your virtual hosts into a separate configuration file called `virtual_hosts.conf`, the directive might look like this:

```
include "C:\website\virtual_hosts.conf"
```

To define virtual directory mappings, use the `Alias` section of the `httpd.conf` file. The virtual directory `iASDemo` is defined here and it will reference files in the physical directory `c:\website\examples`. Note, the ending slash must be a forward slash:

```
Alias /iASDemo/ "c:\website/examples/"
```

Including Directives Based on the Module Loaded

Conditional logic can be used in your configuration files. These conditional directives enable you to include directives if a specific *iAS* module is loaded. If the module isn't loaded, the directive is ignored. For example:

```
<IfModule mod_mime_magic.c>
    MIMEMagicFile conf\magic
</IfModule>
```

Configuring Multiple Sites Using the VirtualHost Directive

Apache can be configured to support multiple Web sites using the `Port`, `Listen`, `NameVirtualHost`, and the `VirtualHost` directives. The ports the sites are using and listening on need to be specified, and they correspond to the ports listed in the `NameVirtualHost` and `VirtualHost` container. `VirtualHost` information can be added using an include file (as previously demonstrated), so virtual hosts' entries can be generated programmatically. This capability of supporting multiple sites, combined

with URL redirection, makes deploying a new release to production a matter of turning on a site just by redirecting the production URL to the new VirtualHost.

The following example shows the *iAS* directives supporting two Web sites, `example1` at <http://localhost:80> and `example2` at <http://localhost:9015>:

```

Port 80
Port 9015
Listen 80
Listen 9015
NameVirtualHost 127.0.0.1:*

<VirtualHost 127.0.0.1:80>
    ServerName localhost
    ServerAdmin admin@localhost
    DocumentRoot "c:/website/example1/htdocs/"
    ErrorLog "c:/website/example1/logs/error.log"
    CustomLog "c:/website/example1/logs/access.log" common
</VirtualHost>
<VirtualHost 127.0.0.1:9015>
    ServerName localhost
    ServerAdmin admin@localhost
    DocumentRoot "c:/website/example2/htdocs/"
    ErrorLog "c:/website/example2/logs/error.log"
    CustomLog "c:/website/example2/logs/access.log" common
</VirtualHost>

```

Using mod_rewrite

`mod_rewrite` is a powerful addition to the Apache Web server. It gives complete manipulation and flexibility over solving URL issues, but this flexibility comes at a cost in complexity and performance. You can use `mod_rewrite` to redirect obsolete URLs to new URLs, redirect failing URLs to another Web server, fix syntax issues like the trailing slash, restrict access to robots, and mitigate content issues.

`mod_rewrite` can be used to address the trailing slash issue, where a user requests a directory instead of a document, such as `/docs/logs` instead of `/docs/logs/`. To fix the problem, direct *iAS* to do an internal and external redirect using a `mod_rewrite` rule, as follows:

```

<Directory "c:\website/example1/docs/logs/">
    Options Indexes FollowSymLinks
    RewriteEngine on
    RewriteBase /logs/
    RewriteRule ^logs$ logs/ [R]
    Order allow, deny
    Allow from all
</Directory>

```

**TIP**

A good article on using `mod_rewrite` can be found at <http://www.engelschall.com/pw/apache/rewriteguide/>.

GUI *iAS* Configuration Editing

Future versions of OEM, included with 9*iAS*, are expected to include the capability to configure Apache, but a graphical configuration tool isn't currently provided by Oracle. Also, we've experienced problems with most of the GUI tools available today, so my recommendation is to use a text editor as previously discussed. Many of the graphical configuration tools are publicized by the Apache GUI-Dev project at <http://gui.apache.org>. The Apache GUI-Dev project was formed to define a common set of goals for developing graphical Apache configuration tools and to publicize information on the tools.

Advantages and Disadvantages of a Graphical Configuration Tool

iAS is designed for high performance and expandability and, consequently, is highly configurable. Because the *iAS* installation doesn't include a graphical configuration tool, several tools have been developed for this purpose. While you might be quite comfortable manually editing the configuration file, others expect that a graphical tool is available to maintain any system file. As with most tools, pros and cons exist to using a graphical tool to maintain an *iAS* installation.

Advantages of a Graphical Configuration Tool

If you're new to *iAS* and, therefore, to Apache or to Web servers in general, you might not be comfortable manually editing the configuration file or you might not know how to implement a certain feature. A graphical interface provides a level of comfort in knowing there's a reduced chance that you'll "break" the configuration. A well-written tool can also guide you in the right direction, providing assistance and recommendations on configuring the directives. By making *iAS* easy to configure and maintain, Apache is more likely to be chosen when comparing and evaluating the Web server choices.

Ideally, a graphical configuration tool enables even expert administrators to configure *iAS* installations more easily than by manually editing the configuration file. By taking all the available resources into account, the tool can reliably recommend the optimal configuration.

If the graphical tool is browser-based, you can manage all the *iAS* servers within the network from a central location. Allowing the user to manage the servers remotely provides an additional level of support because the user doesn't have to be at the physical node where Apache is installed or to connect using remote control software.

Disadvantages of a Graphical Configuration Tool

The main disadvantage of using a graphical configuration tool is it might make administering the installation more time-consuming or difficult. If the tool isn't easy to use or doesn't provide all the required functionality, it might hinder more than it helps.

Another disadvantage is if the tool isn't well written, it might break your configuration. If the configuration file contains directives the tool doesn't recognize, or contains directives where the tool doesn't expect to find them, the tool might not correctly apply the changes. In fact, our experience has shown that most of the tools do, in fact, break the *iAS* configuration files.

A third disadvantage, similar to the disadvantage of a poorly written tool, is the tool must keep up with Apache releases and should work on all platforms supported by Apache. Development of the Apache Web server is extremely active and new releases are published regularly. The tool must understand the directives available for each version and operate dynamically to allow the administrator to configure the directives applicable to the installed version.

Comanche

Comanche is an Apache graphical configuration tool that derives its name from Configuration Manager for Apache. Comanche is available in binary form from Covalent at <http://www.covalent.net/projects/comanche> for Windows NT/2000, Linux, HP-UX, Solaris, FreeBSD, and Irix. In the spirit of the open source movement, the source code is also available. If you want, you can download the Tcl/Tk toolkit and use the source code to customize Comanche to meet your needs.

In experimenting with Comanche, I found that versions 3.0b4 and 2.0b6 are available. Attempting to use 3.0b4 on Windows 2000 resulted in several registry error messages, so I used version 2.0b6 instead.

To run Comanche, simply download and extract the files into the desired directory. Because Comanche is available in executable form, execute the `comanche.exe` file (NT/2000) or `comanche2.0` (UNIX) to start the application.

Managing and Configuring Modules

Managing and properly configuring *iAS* modules is important to provide the fastest possible performance from your *iAS* servers. This section covers the key *iAS* modules.

Configuring the PL/SQL Module (DADs)


The PL/SQL module can be administered via a browser-based configuration. You can also edit the `wdsrv.app` file. Because the browser-based functionality is easy to use and provides all the needed functionality, I recommend using it rather than editing the file. The primary configuration for the PL/SQL module is to administer Database Access Descriptors (DADs) and their specific configurations.

DADs contain the information needed by iAS components to connect to Oracle databases. DADs can be used by any component that supports configurable database access. iAS provides a Web page for creating and configuring DADs.

To set up a DAD in a browser access, use the following URL:

 `http://localhost/pls/admin_/gateway.htm`

In older versions of iAS prior to version 1.0.2.2, use the following URL:

 `http://localhost/pls/simpdedad/admin_/gateway.htm`

In iAS version 1.0.2.2, security was added to the PL/SQL module configuration. These additions were made in the \$ORACLE_HOME/Apache/modplsql/cfg/wdbsrv.cfg file. To edit the PL/SQL module configuration, it's necessary to log in to a valid database schema (such as portal30). Prior versions had a potential security bug that allowed anyone to administer the PL/SQL module through a browser, if they knew the URL.

Adding a New DAD

To add a new DAD, select Gateway Database Access Descriptor Settings, and then select Add Default (blank configuration). Enter values for the following fields:

NOTE

If Database User and Password are omitted, users are prompted for the valid database user ID and password.

- **DAD Name** For example, example1
- **Database User** Also known as schema name (optional)—if you leave this blank, the user is prompted for a valid database user name and password. In other words, implementing database authentication
- **Database Password** (Optional if Database User is left blank)
- **Connect String** The ORACLE_SID
- **Session cookie** Used for Portal, usually left blank
- **Session state** Preserves package/session state between requests, usually set to no
- **Connection pool parameters** Connection pooling improves performance. The default setting turns connection pooling on. Connection pooling saves

your site the expensive processing time of opening and closing database connections

- **Default Home Page** The page to be used if no procedure or package is specified in the URL
- **Document Access Information** Used for uploading and downloading documents in the database
- **Path Aliasing** Map a word to an absolute path

Accessing Your New URL and DAD

After creating the previous example1 DAD, the virtual path to access database packaged procedures is

 `http://localhost/pls/example1/package.procedure`

The components of `http://localhost/pls/example1/dbinfo` are as follows:

- **Protocol** http or https
- **Hostname or IP address** localhost
- **Port** 80 if omitted, default of 80 assumed
- **Module** pls
- **DAD** example1
- **Database Packaged Procedure** package.procedure name

Configuring the Java Module

iAS comes preconfigured with the `mod_jserv` module. JServ has two components: the `mod_jserv` component written in C and the servlet engine written in Java. The `mod_jserv` component serves as an intermediary between iAS in the servlet engine, while the servlet engine contains the servlet application programming interface (API).

JServ

The JServ configuration is located in the `$ORACLE_HOME/Apache/jserv/conf/jserv.conf` that contains the configuration of the JServ engine. A `jserv.properties` file also contains initialization settings for the Java Virtual Machine (JVM).

Java programs are compiled to class files, which can be contained within a Java Archive File (JAR). JServ uses class and jar repositories called *servlet zones*. Each zone can have its own set of classes and each zone can be configured to use a separate JVM.

TIP

For load balancing and improved security, use servlet zones and separate JVMs for large sites with the majority of applications written in servlets.

A number of settings are in the `jserv.conf` file. The directive

```
ApJServManual off
```

tells the JVM to start up when the Apache Web server is started up. If you need to start multiple JVMs for security or load balancing purposes, set the `ApJServManual` directive as follows:

```
ApJServManual on
```

With `ApJServManual` turned on, the JVM needs to start and restart scripts to run the JVMs. In addition, each JVM needs to reside on its own port. `ApJServProperties` specifies the location of the JServ properties file. This directive is ignored in a manual startup.

```
ApJServProperties "C:\Oracle\Ora81\Apache\Jserv\conf\jserv.properties"
```

`ApJServLogFile` specifies the log file for the C portion of `mod_jserv`. In UNIX, you need to make sure the owner of the JVM process has necessary write permissions. Log messages are written to the Apache error log if the directive is set as follows:

```
ApJServLogFile DISABLED
```

To specify a log file, set `ApJServLogFile` using a path and filename, as follows:

```
ApJServLogFile "C:\Oracle\Ora81\Apache\Jserv\logs\mod_jserv.log"
```

`ApJServLogLevel` can be set to any of the following levels: `debug`, `info`, `notice`, `warn`, `error`, `crit`, `alert`, or `emerg`. The default level is `notice`. If you change this to `debug` or `info`, and then restart Apache, you see the following messages in the `mod_jserv.log`:

```
ApJServLogLevel info
[02/06/2001 03:25:29:695] (INFO) wrapper: Shutdown done (PID=256)
[02/06/2001 03:25:29:695] (INFO) Apache Module was cleaned-up
[02/06/2001 03:25:30:206] (INFO) wrapper: Java Virtual Machine started (PID=588)
[02/06/2001 03:25:30:206] (INFO) wrapper: controller started (PID=256)
```

`ApJServDefaultHost` specifies the host running the JVM. To have the JVM run on a different host than the Web server, specify `ApJServManual on` and start the JVM manually.

ApJServSecretKey enables you to specify a key file that enables the administrator to restrict access to the servlet engine.

```
ApJServSecretKey C:\Oracle\Ora81\Apache\Jserv\conf\jserv.secret.key
```

If ApJServSecretKey is disabled, any process on the machine can connect to the servlet engine and execute servlets.

```
ApJServSecretKey DISABLED
```

ApJServMount sets the path mapping to a servlet zone. This directive can be set more than once, so multiple relative paths can point to the same location.

```
ApJServMount /servlets /root
ApJServMount /dev /dev
```

To test if JServ is working, reference the following URL, which calls the Java servlet IsItWorking:

```
http://localhost/servlets/IsItWorking
```

The code for the IsItWorking servlet is located in the \$ORACLE_HOME/Apache/Jserv/Servlets directory.

To compile your own Java servlet, set your environment PATH variable to include

```
$ORACLE_HOME/Apache/jdk/bin
```

and set your CLASSPATH variable to include the following JAR file:

```
$ORACLE_HOME/Apache/Jsdk/lib/jsdk.jar
```

The following code sample creates a servlet using a “Hello World” example:

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class HelloWorld extends HttpServlet {
    public static final String MSG = "Hello World, mod_Jserv is Cool!";
    public void service (HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException
    { response.setContentType("text/html");
      PrintWriter out = response.getWriter();
      String server = getServletConfig().getServletContext().getServerInfo();
      out.println("<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2//EN">"
        + "<HTML> <BODY BGCOLOR=\"#FFFFFF\">"
        + " <H1>" + MSG + "</H1>");
```

```
        + "</BODY> </HTML>");  
    }  
}
```

Save the file as HelloWorld.java and compile the code using the following syntax:

```
javac HelloWorld.java
```

Now to execute your HelloWorld servlet, use the following URL:

```
http://localhost/servlet/HelloWorld
```

ApJServMountCopy specifies if virtual hosts inherit the primary mount points, allowing the administrator to set up a shared mount point for servlets. This applies only to sites using virtual hosts.

```
ApJServMountCopy on
```

ApJServAction maps a file extension, such as .jhtml or .xml, contained in a URL to a servlet. For example, when I set the following in the jserv.conf file:

```
ApJServAction .jhtml /servlets/HelloWorld
```

When a document request arrives with the jhtml extension, the HelloWorld servlet is invoked.

```
http://localhost/tryme.jhtml
```

Servlet Engine

The mod_ose module enables *iAS* to communicate with the OSE by using the Net8 protocol, HTTP tunneling. It benefits from the Net8 features load balancing, firewall support, and connection manager.

Servlets benefit from the mod_ose behavior of maintaining state between requests. State is maintained for the duration of the *iAS* process. This behavior differs from mod_jserv, which doesn't maintain state between requests.

The configuration file is located in \$ORACLE_HOME/Apache/Apache/conf/mod_ose.conf file.

The default configuration file, httpd.conf, references the file oracle_apache.conf, which references the file mod_ose.conf. In httpd.conf, a file include also exists for oracle_apache.conf.

Servlet Engine Properties The jserv.properties file contains initialization settings for the JVM. This file location is set in the jserv.conf by the ApJServProperties directive:

```
ApJServProperties "C:\Oracle\Ora81\Apache\Jserv\conf\jserv.properties"
```


When this file is updated, the JVM must be restarted to reflect the changes. The `wrapper.bin` directive provides the path for the JVM executable and is used only in automatic mode, which is the default.

```
wrapper.bin=C:\Oracle\Ora81\Apache\jdk\bin\java.exe
```

The `wrapper.bin.parameters=[parameters]` property contains parameters passed to the JVM and is used only in automatic mode. `Wrapper.bin.parameters` can have multiple values.

The `wrapper.path=[path]` is used to pass the PATH environment to the JVM. On NT/2000, delimit the statement with semicolons. On UNIX, delimit it with colons. `Wrapper.path` is set in the default configuration.

The `wrapper.classpath=[path]` property is used to set the CLASSPATH environment variable for the JVM to use. The classes you want automatically reloaded on modification cannot be in this class path or the class path of the shell in which iAS is started.

The default settings for `wrapper.classpath` are as follows:

```
wrapper.classpath=C:\Oracle\Ora81\Apache\jdk\lib\tools.jar
wrapper.classpath=C:\Oracle\Ora81\Apache\Jserv\ApacheJServ.jar
wrapper.classpath=C:\Oracle\Ora81\Apache\Jsdk\lib\jsdk.jar
# The following classpath entries are required to run EJBs
wrapper.classpath=C:\Oracle\Ora81\lib\aurora_client.jar
wrapper.classpath=C:\Oracle\Ora81\lib\vbjorb.jar
wrapper.classpath=C:\Oracle\Ora81\lib\vbjapp.jar
```

The `wrapper.env=[name]=[value]` property is used to pass environment variables to the JVM in name-value pairs. The default value is the bin directory for iAS:

```
wrapper.env=PATH=C:\Oracle\Ora81\bin
```

The `wrapper.env.copy=[name]` property copies environment variables for specified names. It can have multiple values.

If set to true, the `wrapper.env.copyall=[true|false]` property copies all environment variables to the JVM.

When debugging Java servlets, use the `log` option to have the servlet write events to the `jserv.log` file. For performance reasons, set `log=false` in a production environment. For example, in a development environment:

```
log=true
```

The previous setting generates log messages similar to the following:

```
[09/08/2001 15:41:52:703 PDT] IsItWorking: init
[09/08/2001 15:43:13:889 PDT] IsItWorking: destroy
```

The location and name of the log file is set using the following directive:

```
log.file=C:\Oracle\Ora81\Apache\Jserv\logs\jserv.log
```

The other directives affecting logging JServ events are

```
log.timestamp=true  
log.dateFormat=[dd/MM/yyyy HH:mm:ss:SSS zz]
```

Logging is processed by a minimum priority thread. If this thread doesn't run periodically, the log queue can overflow, resulting in an out-of-memory error. The two directives that mitigate this condition are `log.queue.maxage` and `log.queue.maxsize`. *Maxage* defines the maximum time a message can stay in the queue, while *Maxsize* defines the maximum number of messages in the queue.

Servlet Zones

Configuration for servlet zones can be found in the `jserv.properties` file. The `port` directive specifies the port iAS JServ listens on. The default is 8007.

The `zones` directive contains a list of servlet zones separated by commas. For example:

```
zones=root,dev
```

Each servlet zone can have its own configuration, which is specified in the properties file.

```
root.properties=c:/Oracle/Ora81/Apache/Jserv/servlets/zone.properties  
dev.properties=c:/website/example1/app/dev.properties
```

The servlet zone has a properties file named `zone.properties`. The `zone.properties` file contains the following directives. The `repositories` property sets the location for the servlets related to this servlet zone, as follows:

```
repositories=c:/Oracle/Ora81/Apache/Jserv/Servlets  
repositories=c:/website/example1/app
```

To enable a servlet class for autoreloading (if the servlet changes), use the `autoreloading.classes` directive. The default value is `True`.

```
autoreload.classes=true
```

To enable servlet-resourced autoreloading, such as properties, use the `autoreload.file` directive. The default value is `True`.

```
autoreload.file=true
```

The *dev.properties* file specifies the servlet repository, as well as classLoader parameters, such as `autoreload.classes`, `init.timeout`, and `destroy.timeout`. The `init.timeout` parameters specify the number of milliseconds allowed for initializing a servlet. The `destroy.timeout` file specifies the time allowed for destroying a servlet.

```
autoreload.classes=true
init.timeout=10000
destroy.timeout=10000
```

The properties file contains session parameters, such as `session.useCookies`, which tells the servlet to maintain session state with cookies, as follows:

```
session.useCookies=true
```

To prevent hanging servlet sessions, set the `session.timeout` to a large number. For example:

```
session.timeout=1800000
```

To determine how often to check for sessions that have timed out, set the `session.checkFrequency` parameter, as follows:

```
session.checkFrequency=30000
```

Containers For J2EE (OC4J)

As of *iAS* version 1.0.2.2, OC4J was not installed and configured in the *iAS* installation. To test your installation of OC4J, execute the following commands from the command line:

```
cd $ORACLE_HOME/j2ee/home
java -jar orion.jar
```

The default configuration file is in `$ORACLE_HOME/j2ee/home/config/server.xml`. Using the `-config` switch, however, you can specify a different configuration file.

```
java -jar orion.jar -config /your_path/server.xml
```

Servlets written for the Tomcat servlet engine should work under the OC4J because Oracle certified OC4J as 100 percent compatible with Tomcat.

Accessing Non-Oracle Databases in Java Type 4 JDBC drivers from Merant can be used to access DB/2, SQLServer, Sybase, and Informix. While these drivers don't ship with *iAS*, they have been certified under *iAS* as of version 1.0.2.2.

Java Development Kit (JDK) Version Oracle recommends using the J2 SDK version that ships with *iAS*. For *iAS* version 1.0.2.2, this is version 1.2.2_07 of the J2 SDK.

Configuring CLASSPATH The CLASSPATH environment variable is configured automatically to run OC4J. Java JAR and class files are loaded directly from the lib directory.

Configuring *iAS* Security

iAS supports both simple and easy-to-configure security solutions, from directory and digest security, to the robust and complex security using secure sockets layer (SSL).

IP and Domain Restriction

The advantage of IP and Domain Restriction is it doesn't require password maintenance. The disadvantage is its vulnerability to spoofing—where intruders falsify the IP address—to gain access to your site. IP Restriction is susceptible to spoofing, but Domain Restriction is more difficult to spoof. Under IP Restriction, requests must be issued from a specified IP address or group of IP addresses designated in the httpd.conf file.

To implement IP or Domain, use the Order, Allow, and Deny directives in the directory container. You can allow or deny from hosts, IP addresses, partial domain name, and network/netmask pair.

```
<Directory "c:\website\example1\htdocs">
    Order Deny, Allow
    All from 172.21.24.0/255.255.0.0
</Directory>
```

TIP

Use Deny before Allow to provide higher levels of security.

Directory Indexing

Use the following directives to make a list of the files in a directory available to your visitors. The Options Indexes MultiViews tells Apache to display contents of a directory if the default page, such as index.htm or index.html, isn't located in the directory.

```
<Directory "C:\website\example1\htdocs\indexed_directory">
    Options Indexes MultiViews
    AllowOverride None
    Order allow, deny
```

```

Allow from all
</Directory>

```

Figure 3-2 shows how Options Indexes Multiviews enables visitors to see files listed in a directory.

.htaccess

The *.htaccess* files often contain authorization information that should be kept from users. *.htaccess* files are great for administration of user access to certain files and directories without having to restart Apache. The *.htaccess* files are parsed for every request. Unfortunately, this additional parsing can cause performance problems, so the following performance section recommends you turn off this functionality. To prevent clients from viewing *.htaccess* files, edit *httpd.conf* to add the following directives:

```

<Files ~ "^\.ht">
    Order allow, deny
    Deny from all
</Files>

```

On NT/2000, *.htaccess* isn't a valid filename, so you must use a name like *_.htaccess*.

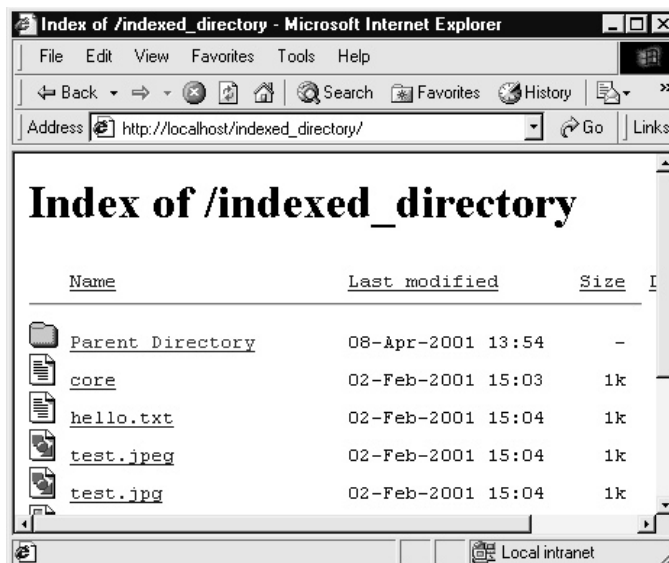


FIGURE 3-2. Directory Indexing in browser

Basic and Digest Authentication

Basic Authentication enables you to assign passwords to users, define groups of users, and assign users to groups. These groups can be assigned to specific files and virtual paths. When Apache receives a request for a file or directory protected with basic authentication, the requestor is required to provide a user name and a password to gain access. Basic Authentication sends unencrypted passwords across the network, making this an insecure method of security. The utility to create and maintain the password file is `htpasswd`. This program can be found in the `$ORACLE_HOME/Apache/Apache/Bin` directory.

The following command creates an authentication file named `password.file` and prompts for a password for the first user name (`scott`). The next command adds an authentication record (user name `student` and password of `mypassword`) to the file named `password.file`. The `-p` parameter tells `htpasswd` to add the password to file, but don't encrypt it—in other words, store it in plain text.

```
htpasswd -c password.file scott
htpasswd -pb password.file student mypassword
```

htpasswd usage:

```
htpasswd [-cmdps] passwordfile username
htpasswd -b[cmdps] passwordfile username password
htpasswd -n[mdps] username
htpasswd -nb[mdps] username password
```

`htpasswd` options:

- **-b** Use the password from the command line, rather than prompting for it
- **-c** Create a new file
- **-d** Force CRYPT encryption of the password
- **-m** Force MD5 encryption of password (default)
- **-n** Don't update file; display results on stdout
- **-p** Don't encrypt password (store in plain text)—note, this only works on NT/2000

By default, `htpasswd` automatically stores the password in the MD5 format (encrypted password). You can then add directives to your `httpd.conf` file or an `.htaccess` file. I strongly recommend you store passwords in the MD5 format, instead

of in unencrypted format. The following directives would be added at whatever level you want to perform basic authentication using the password file. In other words, if you place these directives at the `VirtualHost` level, they apply to the entire virtual host. If you place these directives at the `Directory`, `File`, or `Location` level, they apply to that specific `Directory`, `File`, or `Location`.

```
AuthName 'Registered'
AuthType Basic
AuthUserFile 'C:\website\example\password.file'
require valid-user
```

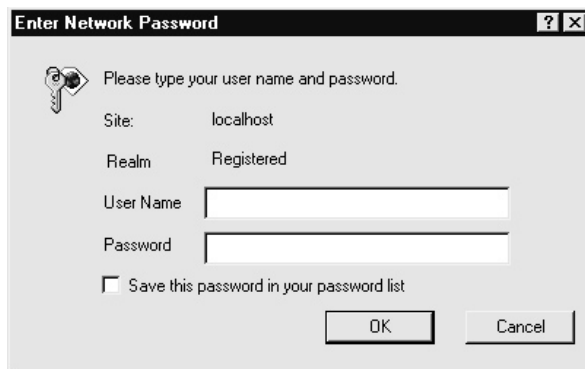
Digest Authentication is similar to Basic Authentication, except it sends passwords encrypted across the network using a cryptographic checksum. The primary difference is you use the `htdigest` program to create a Digest Authentication file. The `htdigest` program doesn't allow passwords to be passed on the command line. You will be prompted (and reprompted) for the password. The use for `htdigest` is

```
htdigest [-c] passwordfile realm username
```

Where:

The `-c` flag creates a new file.

Once basic or digest security is configured, your Web site visitors are greeted by a screen, as shown in the following illustration, prompting them for a user name and a password.



If the incorrect user name and password are entered three or more times, or if they select the `Cancel` button, the user is presented with an `Access Denied` message in his browser, shown next.



Anonymous

Anonymous access, often used on FTP sites, can be implemented by adding Anonymous directives to the .htaccess file. Anonymous access is often accompanied by prompting for a valid e-mail address on connection to the server. Anonymous directives list valid user names. Anonymous_LogEmail requires an e-mail address to be entered for the password. Anonymous_VerifyEmail states whether the server syntactically checks for valid e-mail addresses containing a dot and @ sign. For example, the following directives can be added to the context of your choice (for example, Global, Directory, File, Location):

```
Anonymous_Authoritative off
Anonymous guest anonymous
Anonymous_LogEmail on
Anonymous_VerifyEmail on
```

Database Authentication

Database authentication uses a database schema (that is, a user name and a password) to authenticate users. By combining database authentication with logic in your application, you can provide application-level authentication. By using the user pseudocolumn, your application logic can use database roles or table-based roles to determine a user's level of functionality. Typically, you build PL/SQL packages to support the level of application authentication you want to provide for your users.

Database authentication is recommended for a small number of users or small intranet Web sites. For large sites or Internet sites, you probably want to use the Single Signon functionality or application logic for security.

**TIP**

When creating a DAD, if the user name and/or password aren't entered, database authentication is enabled for that DAD.

SSL

Secure Sockets Layer (SSL) is the encryption protocol that handles HTTPS (HTTP over SSL). SSL isn't normally packaged with Apache, however, Oracle has thankfully packaged this module into the iAS version of Apache. SSL uses encryption called *public key cryptography*. The server sends the client a public key. The client encrypts data using the public key and sends it to the server, which can decrypt the data using the private key. SSL is the recommended security implementation for supporting e-commerce sites.

**TIP**

Use SSL for security of Internet applications from end-to-end.

Oracle has provided a demo certificate to be used in the development environment. In a production environment, you want to create a new key and certificate.

Steps to Create a New Key and Certificate

1. Create a new directory called demoCA.

```
mkdir c:\demoCA
```

2. Copy the openssl configuration file into a demoCA directory.

```
copy $ORACLE_HOME\apache\open_ssl\bin\openssl.cnf c:\demoCA
```

3. In the Command window, change directory to demoCA and set the path to include c:\oracle\ora81\apache\open_ssl\bin.

```
SET PATH=$ORACLE_HOME\apache\open_ssl\bin;%PATH%
```

4. Edit the openssl file and comment out the following lines:

```
#RANDFILE= $ENV::HOME/.rnd
#oid_file= $ENV::HOME/.oid
```

5. Replace the line starting with dir:

```
dir= c:\demoCA
```

Steps to Creating a Key

1. Create a random key by typing:

```
openssl genrsa -des3 -out priv.key 1024 -config
```

2. Create a new certificate by typing:

```
openssl req -new -key priv.key -out certreq.csr
-config c:\demoCA\openssl.cnf
```



NOTE

When openssl prompts you for the CommonName, enter your host.domain.com.

3. You should now have the files priv.key and certreq.csr in the demoCA directory.
4. Go to **www.verisign.com**, select Get a Server Certificate, and then select the Trial version.
5. Follow the instructions and fill out all information accurately.
6. Open certreq.csr with the editor and copy the encrypted message into the text box provided by Verisign.
7. Verisign will e-mail the SSL trial certificate.
8. Save the e-mail to a text file.
9. Open this text file and delete text appearing before:

```
---BEGIN CERTIFICATE---
```

10. Save this as cert.crt file in c:\demoCA directory.

11. Remove the password from the key file.

12. Copy the private key and register the key.

```
copy priv.key priv.key.org
openssl rsa -in priv.key.org -out priv.key
```

13. Edit your httpd.conf file, and then locate and replace the following two entries:

```
SSLCertificateFile conf\ssl.crt\server.crt
```

with

```
SSLCertificateFile c:\demoCA\cert.crt SSLCertificateKeyFile
```

and replace:

```
conf\ssl.key\server.key
```

with

```
SSLCertificateKeyFile c:\demoCA\priv.key
```

14. Using the public and private key Oracle provided, test the SSL functionality. Modify the httpd.conf file's VirtualHost container. For example:

```
<VirtualHost 127.0.0.1:443>
DocumentRoot c:\website/example1/htdocs/
ServerName localhost
SSLEngine on
SSLCertificateFile conf\ssl.crt\server.crt
SSLCertificateKeyFile conf\ssl.key\server.key
Listen 443
</VirtualHost>
```

15. Restart Apache and in your browser go to **https://localhost:443/**. If SSL is working, you should see the standard Web page for your localhost server.



TIP

Secure log files and directories from read/write access. Log files contain private information about who is accessing your Web site and how they're using it.

Monitoring Your iAS Web Server

Knowing how to monitor your iAS Web Server can help you achieve better performance, improved security, and reliability. Monitoring can also help you understand how your users are using the Web site.

Monitoring Error and Log Files

Excessive or repeated errors consume the system's resources and slow response times. Check the log files regularly to detect inefficiencies in the iAS configuration.

For example, an image map with undefined areas might cause a repeated HTTP Error 500. Every user who clicks one of these areas generates the error. You can eliminate the problem by eliminating the "holes" in the image map. Another example is a repeated HTTP Error 404, which is usually caused by a broken hypertext link. Any repeated error slows down your entire application.

Regular monitoring of log files provides a good understanding of how users access your site. Use this information to optimize allocation of hardware and server resources. You can use tools, such as WebTrends, for detailed site use monitoring.

Don't allow your log files to grow without limit. When a log file reaches a certain size (the size greatly depends on the size of your site, but 1–5MB is a good standard rule), archive the file and start a new log file. You need to stop Apache to accomplish this. This avoids the overhead of writing to a large file.

Error Log

Apache uses the `mod_log_config` module to provide flexibility in terms of logging. `ErrorLog` specifies the location of the `error.log`. The default location for the error log file is `$ORACLE_HOME/Apache/Apache/logs/error_log`, but log files can be located wherever you specify. The error log file, unlike some other log files, cannot have a customized format. Under UNIX, the error log can be redirected to the system log daemon using the following directive:

```
ErrorLog syslog
```

Log Level and Format

Settings for log files can be placed within containers for `VirtualHosts`, so each of your Web sites has its own set of log files. The `LogLevel` specifies the types of errors added to the log file. See Chapter 23 for a listing of log levels. Some log files, like the `access_log`, can have a specified log format. `LogFormat` is the directive that can be used to modify the standard format of the log file. `LogFormat` takes two arguments: the format string and a name for the format (optional). `LogFormat`, along with `TransferLog`, can be defined in the `VirtualHost` container to customize the format for tracking all site visits and activity. In my `VirtualHost`, I have the following specified:

```
TransferLog c:/website/example1/logs/access.log
LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\
  \"%{User-Agent}i\"" combined
```

TIP

Excessive logging can cause performance problems. Set logging level and log files to capture only what your site needs.

To set the logging level to capture warnings (or worse), use the `LogLevel` directive, as follows. The default is `warn`, as discussed in the following tuning section, using `LogLevel error` can improve *iAS* performance by over 150 percent. If you require more detailed logs for analysis of your user community (see Chapter 23 for more information about log file analysis), however, you won't receive detailed logs if `LogLevel` is set to `error`.

```
LogLevel warn
```

Monitoring Processes and Memory Use

To monitor memory use on UNIX, various utilities depend on the flavor of UNIX. `top`, `sar`, and `vmstat` are all useful tools. The `ps -aef` command is useful to determine

active processes in UNIX. On NT/2000, the Task Manager can provide details on process and memory use.

iAS mod_status and mod_info

iAS mod_status generates server status information when enabled. To enable the server status information to be available only by clients accessing the site from LocalHost, add the following directives to `httpd.conf`. `mod_status`, by default, presents standard status information to obtain additional server status information.

```
ExtendedStatus on
```

The following directives are used with `mod_status`:

```
<Location /server-status>
    SetHandler server-status
    Order deny,allow
    Deny from all
    Allow from localhost
</Location>
```

To access server status information in the browser, use the link <http://localhost/server-status>, as you see in Figure 3-3.

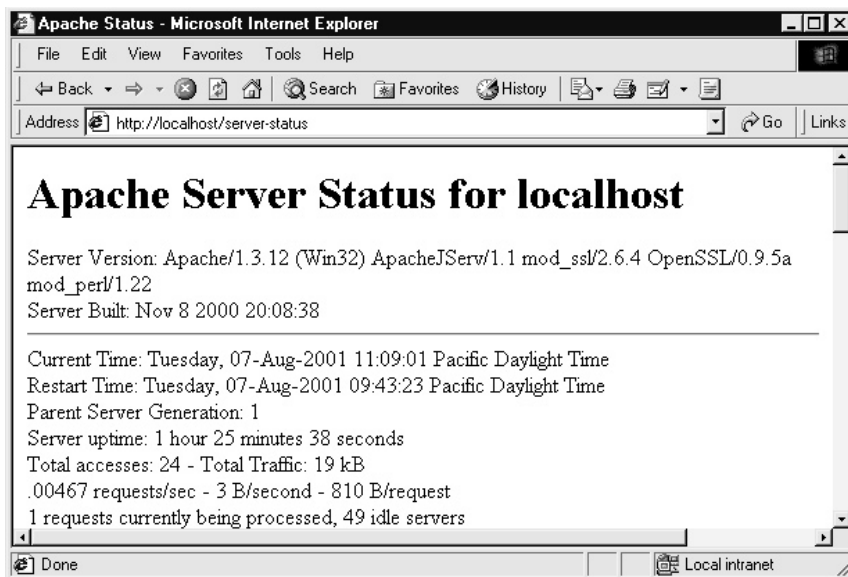


FIGURE 3-3. Apache (iAS) Server Status

TIP

Server status information can be used by hackers, so it's best to place the allow and deny directives in the location container where server-status is turned on.

To obtain module configuration information, showing configuration information such as what modules are loaded, use the `mod_info` module. Turn on `mod_info` by adding the following directives to `httpd.conf`:

```
<Location /server-info>
    SetHandler server-info
    Order deny,allow
    Deny from all
    Allow from localhost
</Location>
```

To access server status information in a browser, use the link <http://localhost/server-info>, which displays a page similar to Figure 3-4.

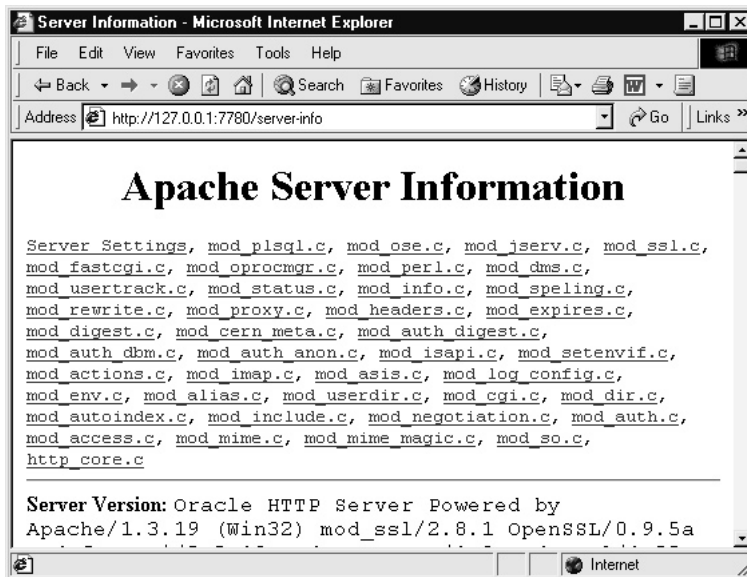


FIGURE 3-4. Apache (iAS) Server Information

Monitoring mod_jserv

mod_jserv has its own set of log files and status information page. In the jserv.conf file—located under \$ORACLE_HOME/Apache/Jserv/conf—uncomment the following code to enable the JServ status page:

```
<Location /jserv/>
  SetHandler jserv-status
  order deny,allow
  deny from all
  allow from localhost
</Location>
```

To see the status page similar to Figure 3-5, restart Apache and, in the browser, go to **http://localhost/jserv/**.

Select the configured hosts link on the status page to view the JServ settings for your Web site, which executes the following URL: **http://localhost/jserv/status?module=127.0.0.1** and displays a page similar to Figure 3-6.

If you select Mapped Servlet Engines, and then select the information on the current status of the servlet engine and the root servlet zone, which executes the following URL **http://localhost/jserv/engine/0/**, you see a page similar to Figure 3-7.

When you select the current status of the servlet engine, you can also see information on the location of the log file and max connections, as in Figure 3-8.

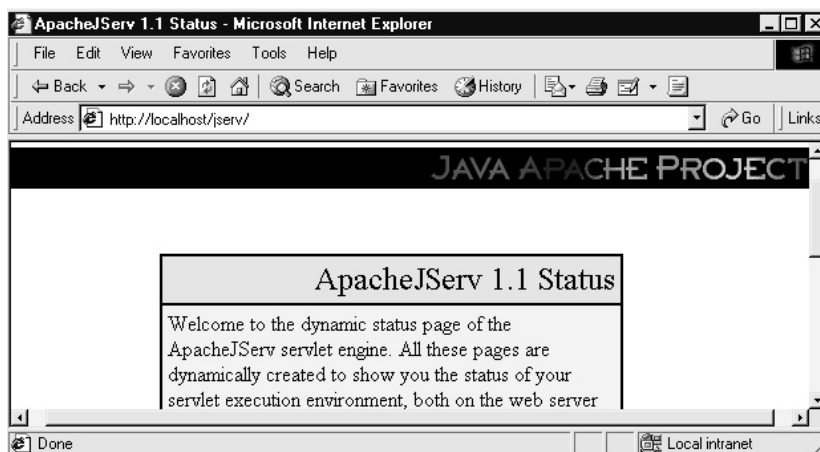


FIGURE 3-5. Apache JServ Status

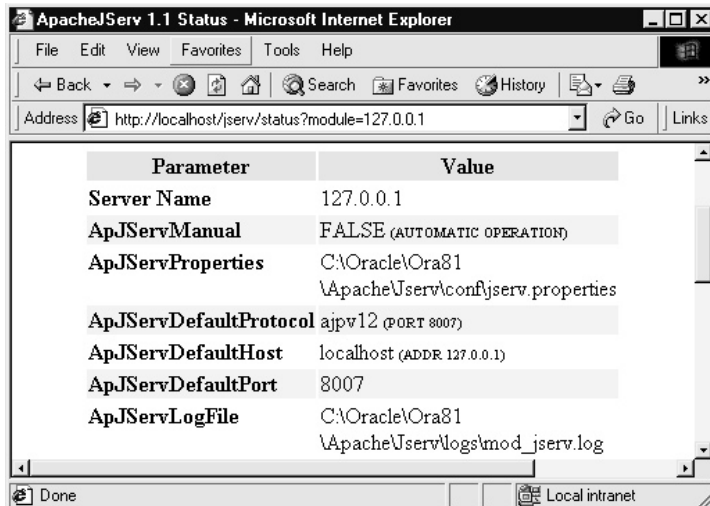


FIGURE 3-6. *Configured Hosts Parameters*

When selecting status on a servlet zone, you can determine whether autoreload is turned on and see the settings for various timeouts, as shown in Figure 3-9.



FIGURE 3-7. *JServ Engine Status*

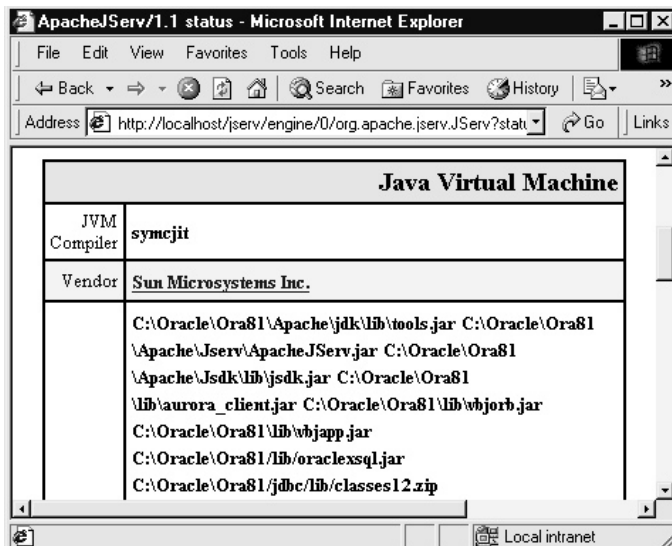


FIGURE 3-8. Java Virtual Machine details

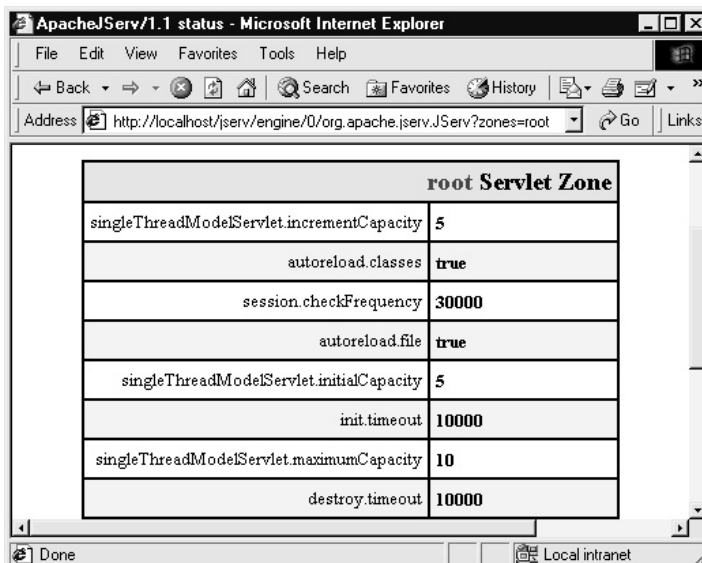


FIGURE 3-9. Servlet Zone information

Tuning iAS

The following tuning tips can help enhance your iAS performance.

Host name lookups occur when iAS performs a Domain Name Server (DNS) lookup on an IP address of the request. These lookups are slow and should be avoided using the directive `HostNameLookup off`. Also, when configuring the `VirtualHost` directive, use the IP address of the server versus the name of the server to prevent DNS lookups.

Use the `Options FollowSymLinks` directive, rather than `SymLinksIfOwnerMatch`, which causes Apache to evaluate whether the owner of the link is the owner of the server and also causes Apache to check the full path of the symbolic link. This checking takes processing time and should be avoided.

`.htaccess` files can cause Apache to run slower because, for every request, the `.htaccess` files in the document directory accessed are parsed. `.htaccess` files can be ignored using the `AllowOverride` directive.

```
<Directory "C:\website/example1/htdocs/">
    AllowOverride None
</Directory>
```

`MinSpareServers`, `MaxSpareServers`, and `StartServers` aren't as important for tuning with the latest versions of Apache. Apache has a robust algorithm for allocating and destroying server processes. `StartServers` (the number of processes Apache creates on startup) is controlled dynamically and, therefore, shouldn't be set. `MinSpareServers` sets the lowest number of processes available. `MaxSpareServers` specifies the highest number of processes available.

Set `MaxClients` (default 256) to a lower level if your site is reaching this level and the clients connecting are experiencing performance issues. `MaxClients` sets the maximum processes allowed on the server. If an HTTP request cannot be satisfied because of this maximum being reached, the client gets a message that the server is unavailable.

Disable unused modules to lessen the memory consumed by Apache. For example, if you aren't using Java, there's no reason to have Apache automatically start `mod_jserv` and the JVM.

Avoid These Features

Avoid the following Apache features: `mod_usertrack`, `mod_session`, URL Rewrite, `.htaccess` files, and Extended Status. Session Tracking has a performance cost, which should be carefully weighed before implementing it. This said, modules that facilitate session tracking, such as `mod_usertrack` and `mod_session`, should be avoided. Rewriting URLs, especially rewriting them using complex rules, should be evaluated when looking at performance. To mitigate the cost of the `mod_rewrite`, use it selectively by enabling and disabling it for only the portions of your site or sites that need it. Because `.htaccess` files are parsed for every request, they should be avoided. The

extended status page generated from `mod_status` has a high performance cost because two system time calls are made for each request.

Tuning Your Logging

Logging can be both CPU- and I/O-intensive. If possible, log errors only by setting `LogLevel error`, as follows:

```
LogLevel error
```

TIP

Using `LogLevel error` versus the default `LogLevel warning` can improve performance by over 150 percent.

Tuning Application Modules

Tuning your Web applications can make or break the performance of your Web site. With a variety of applications supported by iAS, a number of tuning options exist. CGI applications can often be tuned by allowing the CGI scripts to be cached. PL/SQL applications can be tuned using tools like SQL trace and Explain Plan. Java database applications can benefit from SQL tuning, as well as class caching.

Tuning CGI/Perl Applications

CGI scripts written in Perl can be cached by using the `mod_perl` module `Apache::Registry`. In the default `httpd.conf` file, the following directives are set to turn on caching for Perl scripts. The `Location` container is used, so I can also not have some CGI scripts cached because caching of CGI scripts can sometimes cause a problem with their execution. This is because of the wrapper Apache uses to enable CGI script caching.

```
Alias /perl/ "c:\website\example1\cgi-bin\"
<Location /perl>
    SetHandler perl-script
    PerlHandler Apache::Registry
    AddHandler perl-script .pl
    Options +ExecCGI
    PerlSendHeader On
</Location>
```

When your URL references `/perl/hello.cgi`, your script is cached. When you reference your script `/cgi/hello.cgi`, however, your script won't be cached. This flexibility is important if your Perl CGI script is using code that can result in abnormalities when cached. To tell if a Perl script is cached or not, check the

mod_perl environment variable, which is always set when handling requests. As demonstrated in Figure 3-10, to generate the status page for mod_perl, add the following directives to httpd.conf and the reference <http://localhost/perl-status/> in your browser:

```
<Location /perl-status>
    SetHandler perl-script
    PerlHandler Apache::Status
    order deny,allow
    deny from all
    allow from 127.0.0.1
</Location>
```

Tuning PL/SQL and PSP Applications

To tune PL/SQL applications and PSPs, build your applications in a way that reduces the load on the database server and performs fast data manipulation language (DML). Tools such as the iAS Database Cache, and mod_plsql features, such as connection pooling, can enable you to reduce the load on the database. Explain Plan, sqltrace, tkprof, and Oracle Expert can help you tune your queries by showing you optimizer

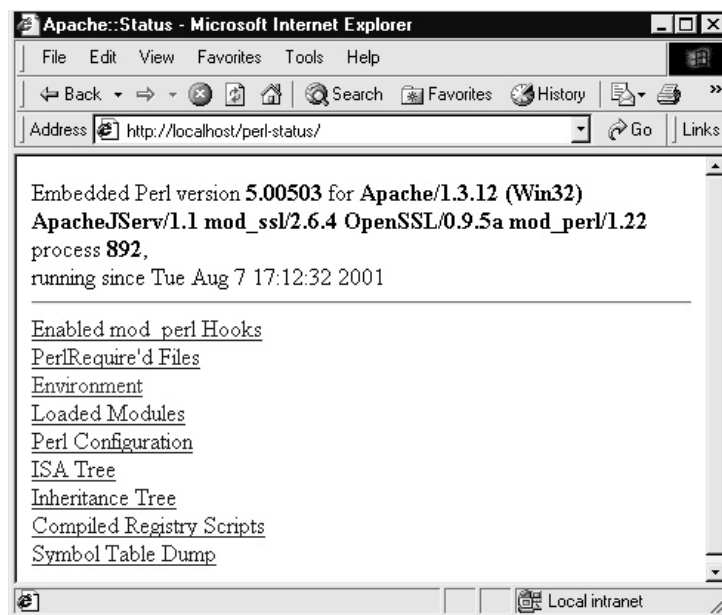


FIGURE 3-10. *Perl status page*

paths and places where you can enhance performance through index creation. Oracle9i provides numerous optimizer hints, new indexing options, and new PL/SQL and database features to improve DML performance.

Use Database Connection Pooling

PL/SQL programs and PSPs can be configured to use database connection pooling. When configuring the database access descriptor, make sure to leave the default setting for connection pooling option on. This tells iAS to pool database connections. Opening and closing database connections causes a big performance drain on the database, so I highly recommend using connection pooling.

9i New Features and Hints

Inserts can be faster (at the expense of some additional disk space use), using the append hint. Inserts, updates, and deletes on tables that don't need to be recoverable can be made faster by setting the affected table to nologging, so no redo log information is generated. Indexes being suppressed by functions in the where clause can take advantage of function-based indexes, which can enhance performance 100 times over.

Tuning Your PL/SQL Code

PL/SQL provides the capability to create PL/SQL tables (that is, arrays). The index of the table is the binary integer ranging from -2147483647 to +2147483647. This table index option is known as *sparsity*, and enables meaningful index numbers, such as customer numbers, employee numbers, and other useful index keys. Use PL/SQL tables to process large amounts of data.

PL/SQL tables can also be used in Oracle Call Interface (OCI) applications, enabling you to pass arrays from OCI into PL/SQL stored procedures to process large batches of data. PL/SQL tables increase performance by operating on tables rather than on single rows.

PL/SQL provides TABLE and VARRAY (variable size array) collection types. The TABLE collection type is called a nested table. *Nested tables* are unlimited in size and can be sparse, so elements within the nested table can be deleted using the DELETE procedure. Variable-size arrays have a maximum size, and maintain their order and subscript when stored in the database. Nested table data is stored in a system table associated with the nested table. Variable-size arrays are designed for batch operations in which the application processes the data in batch array style.

Use Explain Plan, Tkprof, and Oracle Expert to Improve Query Performance

To maximize performance and minimize execution time, optimize application SQL statements. Tuning SQL is often the effort of rewriting SQL statements either to do less work or to improve the parallelization factor of the query.

Explain Plans are crucial to tuning SQL and examining the execution path. The Explain Plan lists the execution path that the Oracle's Optimizer chooses for a given query, enabling you to review the execution path before it's submitted. Use Explain Plan to analyze queries and to tune the query before it's executed. If Explain Plan reveals a poor execution path, you can tune the SQL statement using hints to obtain a better Execution Path.

To use the Explain Plan utility, select permission must be granted on the tables you're attempting to explain. You must also have select and insert permissions on the PLAN_TABLE table. To create the Explain Plan table, run the \$ORACLE_HOME/rdbms/admin/utlxplan.sql script.

You can also generate a SQL trace file and use it with the *tkprof utility*, which shows the different phases, including parsing, fetching, and execution. SQL trace can be turned on for an individual session or for the entire database. To turn on SQL trace for your current session, execute the data dictionary language (DDL) statement in a SQL*Plus window.

```
ALTER SESSION SET SQL_TRACE = true;
```

To turn on sqltrace for the entire database, modify the init.ora file, adding sql_trace = true, and then restart the database.

Oracle Expert provides a good analysis of DML running on your database and can generate a report on indexes it recommends creating and removing, based on the DML collected and analyzed. Oracle Expert comes with the Oracle Tuning Pack.

Caching PL/SQL Caching PL/SQL calls eliminates additional network round trips to the database server. Caching is used by Portal. To invoke caching, make calls to OWA_CACHE package.

Tuning JSP/JServ Applications

When tuning JServ applications, consider preloading commonly accessed classes at JServ startup, which is similar to pinning PL/SQL packages in a database. Add classes to pin in the servlets.startup directive in the zone properties file.

Use Connection Pooling and Connection Caching

JDBC connections are expensive in terms of performance and, therefore, should be avoided by using connection pooling. JDBC allows physical database connections to be shared or pooled through logical connections. A *logical connection* is one in which the JDBC client borrows the physical connection and returns the physical connection when it closes the logical connection. This model has been enhanced with the Oracle extension to JDBC 2.0 that introduces database connection caching. Connection caching allows for improved connection management by associating a

number of pooled logical connections with a physical database connection containing a database schema name.

Use Autoreload Only in Development, Not Production

Autoreload is an option commonly used in development because it enables you to change classes without restarting the Apache server. In a production environment, Autoreload can cause poor performance because it requires the classes to be checked for changes for each call to JServ.

```
Set Autoreload.classes = false
Set Autoreload.file = false
```

Use the Single-Threaded Model

Using the *single-threaded model* can improve application performance by more than 25 percent. This gain is the result of the decrease in synchronization bottlenecks. The single-threaded model enables you to have a pool of serially reusable servlets, which are thread safe. To implement the model, establish the servlet connection in the `init()` method and disconnect from the database server in the `destroy` method. Use the following settings when configuring applications to use a single-threaded model:

```
single ThreadedModelServlet.initialCapacity
single ThreadedModelServlet.incrementCapacity
single ThreadedModelServlet.maximumCapacity
```

Load Balancing JServ

Starting multiple JServ processes can benefit even a single processor machine because Apache won't automatically restart the process if it dies. *Load balancing* can increase the performance of Java classes that have synchronized methods by making parallelism possible. A JVM process can go down when a servlet terminates with a `System.exit()`, so load balancing provides reliability through redundancy. Starting multiple JServ processes allows a lot of flexibility for your site because servlet zones can be distributed on different JServ. JServ processes can be started with a different user ID than that which owns Apache. Every JServ process can be started with a unique CLASSPATH and JDK version, making it possible for sites to have some servlets that need various versions of the JDK running in their own environment. An additional script is required to start up more than one process.

JServ, by default, is automatically started each time Apache is started with `ApJServManual` set to `auto` (the default setting) in `jserv.conf`.

```
ApJServManual auto
```

To start JServ manually, turn the manual setting on with

```
ApJServManual on
```

Manually Start the JServ Processes in UNIX

1. Stop Apache Web server.

```
apachectl stop
```

2. Modify jserv.conf to set ApJServDefaultPort to a port other than default 8007.
3. Modify jserv.conf to set ApJServManual on.
4. Modify jserv.properties to set the old port number to the new port number specified for ApJServDefaultPort.
5. Start Apache Web server.

```
apachectl start
cd $ORACLE_HOME/bin
```

6. Create a new file called jservctl.

```
#!/bin/ksh
# This shell script manually starts the JVM.
# It is a simplified script based
# on a script found on Metalink
# http://metalink.oracle.com Note: 123533.1
function usage {
    echo " "
    echo "jservctl usage:"
    echo "To start: jservctl "
    echo "To stop: jservctl -s "
    echo "To see version: jservctl -v"
    echo "To see options: jservctl -V"
    echo "To get usage: jservctl -h"
    echo " "
}
echo "1 is $1"
if [[ "${1}" = "0" ]] then
    usage
    exit -1
fi
if [[ "${1}" = "-h" ]]
then
    usage
    exit 0
fi
JSERV_CMD=$1
APACHE_HOME=$ORACLE_HOME/Apache
```



```

JDK_HOME=$APACHE_HOME/jdk
JSDK_HOME=$APACHE_HOME/Jsdk
JSERV_HOME=$APACHE_HOME/Jserv
JSDK_JAR=$JSDK_HOME/lib/jsdk.jar
JSERV_JAR=$JSERV_HOME/libexec/ApacheJServ.jar
JSERV_NAME="org.apache.jserv.JServ"
JSERV_LOG=$JSERV_HOME/logs
JSERV_CFG_DIR=$JSERV_HOME/etc
CLASSPATH=$CLASSPATH:$JSDK_JAR:$JSERV_JAR
export CLASSPATH
LOG=$JSERV_LOG/jserv.log
PROPERTY=$JSERV_CFG_DIR/jserv.properties
CLASSES=$CLASSPATH
$JDK_HOME/bin/java -classpath $CLASSES $JSERV_NAME
$PROPERTY $JSERV_CMD &> $LOG


```

JServ and Session Tracking JServ uses session cookies to bind a session with a particular JServ process. When a request arrives, `mod_jserv` randomly chooses to which JServ process to route the request and adds a cookie trailer specifying the `ApJservRoute`, which is added to the session cookie before satisfying the client request. Subsequent requests are routed to the appropriate JServ process, based on the cookie value for `ApJservRoute`.

Turn Off Debug and Developer Flag When Deploying JSPs to Production

In the servlet zones properties file, there's a setting for the init args for developers to turn on debugging. The *Debug* mode is used to generate debugging messages. The *Developer* mode is used to specify whether or not at run time the JSP should automatically recompile and reload any JSPs that have changed since they were last loaded. The default setting for Developer mode is True.

Use the following directive to turn off Developer mode and Debug mode:

```
 servlet.oracle.jsp.JspServlet.initArgs=debug=false, developer_mode=false
```

Hardware and Operating System Tuning

You can use the operating system and Web utilities to analyze *iAS* performance, including process CPU time and memory use, and connection and request statistics. A number of other ways exist to monitor the performance of your applications, servers, and sites. The following sections cover the general tuning items.

Reviewing Hardware Recommendations

In addition to the following general recommendations, make sure your hardware resources are adequate for the requirements of your specific applications. To avoid hardware-related performance bottlenecks, each hardware component should operate at no more than 75 percent to 80 percent of capacity. In the beginning of a system's life, the 80 percent of capacity rule is common sense, but it's often forgotten once the application goes into production. Loading the last 20 percent of the system is what can cause the worst performance problems.

The following is the *absolute* minimum hardware configuration for most *iAS* deployments on the Sun Solaris platform:

- Sun UltraSparc II at 168 MHz or Pentium Pro at 200 MHz
- Memory: 128MB, recommended 1GB
- Network Connection: 100 Mbps
- Disk Space:
 - Minimal Edition: 630MB
 - Standard Edition: 1.35GB
 - Enterprise Edition: 3.60GB

Processor and memory resources should be more than adequate to handle the maximum traffic on your network connections. If your network becomes a bottleneck, you can upgrade to faster network interface cards or install multiple network interface cards on each machine.

Tuning *iAS* on Solaris

To monitor *iAS* performance on Sun Solaris systems, use the `top`, `vmstat`, `sar`, and `ps` utilities to monitor memory use, swapping, paging, and processor use by *iAS* processes.

Oracle recommends installing *iAS* on its own machine for optimum performance and monitoring. Processor use varies with each individual application. If the application is network-intensive, the listener processes consume the majority of processor cycles. If your application is code-intensive, the modules consume the majority of the work, so modules perform the majority of the processes. If the application is database-intensive and the database is on the same box as *iAS* (not recommended for a production site), the database processes consume most of the machine CPU and I/O.

Tuning the Operating System

Sun Microsystems regularly updates the Solaris operating system components (TCP/IP, subsystem). TCP/IP is heavily used by *iAS*, so be sure you've installed the latest patches.

Sun also provides the Solaris Internet Server Supplement, which is a set of add-on modules specially tailored for Solaris systems that host Web sites. Make sure to obtain this supplement from Sun. The following subsections discuss items you can set at the operating-system level for optimum system performance.

File Descriptors Make certain the limit on file descriptors per process is set at the maximum before starting *iAS*, using the `unlimit` command, as follows:

```
$ unlimit descriptors
```

Transmission Control Protocol (TCP) Settings The following table lists the recommended TCP parameters and values for Solaris. These values are good recommendations and haven't changed in a long time, but you might want to contact Oracle support for the most current recommendations.

Parameter	Recommended Value
<code>tcp_conn_req_max_q0</code>	1024
<code>Tcp_conn_req_max_q</code>	1024
<code>tcp_slow_start_initial</code>	2
<code>tcp_close_wait_interval</code>	60000
<code>Tcp_conn_hash_size</code>	32768
<code>tcp_xmit_hiwat</code>	32768
<code>tcp_rcv_hiwat</code>	32768

To set these TCP parameters, you must first connect to the root UNIX account (`su root`), and then use the following command, replacing each parameter with the suggested value of the preceding parameter:

```
# /usr/sbin/ndd -set /dev/tcp parameter value
```

Using the netstat Utility Solaris contains a TCP/IP statistic, known as *tcpListenDrop*, that counts the number of times a connection is dropped because of a full queue. You can use the Solaris `netstat` utility (`-s` option) to report networking

statistics, including `tcpListenDrop`. Applications that have high `tcpListenDrop` counts should increase the size of the queue by specifying higher values for the backlog to the listener (the `listen()` call). Set the maximum backlog size by adjusting the value of the `tcp_conn_req_max` parameter higher. In the case of an initial handshake, an incoming packet is sent with only the Synchronized Sequence Numbers (SYN) flag set. When a packet is sent, the server makes an entry in the listen queue, and then sends another packet to acknowledge the first packet. It also includes a SYN flag to reciprocate the synchronization of the sequence number in the opposite direction. The client then sends another packet to acknowledge the second SYN, and the server process is scheduled to return (from the `accept()` call), subsequently moving the connection from the listen queue to the active connection list.

Solaris has two TCP tunable parameters—`tcp_conn_req_max_q` and `tcp_conn_req_max_q0`—which specify the maximum number of completed connections waiting to return from an `accept()` call and the maximum number of incomplete handshake connections, respectively.

Use the `netstat` utility (`-s` option) to monitor TCP statistics and to determine connection-drop activity, as well as the type of drops. The `tcpHalfOpenDrop` statistic is incremented when an in-doubt connection is dropped. The default value for `tcp_conn_req_max_q` is 128 and the default value for `tcp_conn_req_max_q0` is 1,024. The default values are typically sufficient and shouldn't require tuning. By examining the statistics with the `netstat` utility, however, you can determine if the parameters need to be adjusted.

TCP implementations use a congestion window limiting the number of packets that can be sent before an acknowledgment. This is used to improve the startup latency and also helps avoid overloading the network. The TCP standard specifies that the initial congestion window should consist of one packet, doubled up on each successive acknowledgment. This causes exponential growth and may not necessarily be ideal for HTTP servers, which typically send small batches of packets. Solaris provides a `tcp_slow_start_initial` parameter, which can be used to double the congestion window from its default of 1 to 2. This improves transmission throughput of small batch sizes.

Contrary to Solaris, NT/2000 doesn't immediately acknowledge receipt of a packet on connection/start, which results in an increase in the connection startup latency. NT/2000 immediately acknowledges if two packets are sent. The difference between the NT/2000 and the Solaris implementation causes performance discrepancies, or higher response times, when NT/2000 clients are used to connect to Solaris servers with a high-speed or LAN-based network. Set the congestion window on Solaris to 2, using `tcp_slow_start_initial` equal to 2.

In Solaris, the `tcp_conn_hash_size` parameter can be set to help address connection backlog. During high connection rates, TCP data structure kernel lookups can be expensive and can slow the server. Increasing the size of the hash table improves lookup efficiency. The default for `tcp_conn_hash_size` is 256. This

parameter must be a power of 2 and can be set in the `/etc/system` kernel configuration file.

Use the `netstat` utility to monitor the overall network traffic, as well as the network traffic for a given interface, using the `(-k)` option.

Monitoring Processor Use

Gather CPU statistics to determine process utilization. You can also monitor system scalability by adding users and increasing the system workload. Use the `sar` and `mpstat` utilities to monitor process use, as described in the following subsections.

Using the `sar` Utility To determine process use, use the following `sar` command:

```
$ sar -u 1 5
```

When you use the `sar` command, you receive a listing similar to the following:

```
$ sar -u 1 5

SunOS Tuscoco_oas3 5.6 Generic_sun4m

%usr    %sys    %wio    %idle
1        1        0        98
3        5        0        92
8        2        0        90
2        2        0        96
```

The `sar` command (`-u` option) provides the statistics identified in the following table:

CPU Statistics	Description
%usr	Percentage the processor is running in user time
%sys	Percentage of processes running in system time
%wio	Percentage the processor spends waiting on I/O requests
%idle	Percentage the processor is idle

Using the `mpstat` Utility On Solaris, you can also monitor CPU processes using the `mpstat` command, as follows:

```
$ mpstat 1 3
```

The `mpstat` utility is similar to the `sar` command: the first argument to `mpstat` is the polling interval time in seconds. The second argument to `mpstat` is the number of iterations. The `mpstat` utility reports the statistics per processor.

Monitoring the Run Queue

Monitor the *run queue* to determine if processes are waiting for an available processor. Use the following `sar -q` command to monitor the run queue:

```
$ sar -q 2 5
```

The following table describes the statistics shown when you use the `sar -q` utility:

Statistic	Description
Runq-sz	Length of the run queue (processes waiting for CPU)
%runocc	Percentage of time occupied
Swpq-sz	Number of processes that have been swapped out and are now ready to run
%swpocc	Percentage of time occupied

Tuning I/O

Direct input/output (I/O) bypasses the UNIX file system cache and copies the file system-based file data directly into user space. Direct I/O on file systems is similar to raw devices. Solaris 2.6 enables direct I/O to be performed using the `directio()` system call. An application can use the `directio()` system call to perform direct I/O processing on a file. To control whether direct I/O to the file system is forced, use the mount command options: `noforcedirectio` and `forcedirectio`.

Use direct I/O to

- Improve large sequential I/O performance
- Improve performance of large files during file transfers
- Eliminate extra buffer copies and file system cache maintenance
- Reduce CPU consumption

Tuning iAS on Windows NT/2000

On Windows NT/2000 platforms, the available tuning options aren't nearly as extensive as on UNIX platforms. Use the Task Manager and Performance Monitor to assess hardware and operating system performance and to verify the application processes being used by *iAS*. The bottom line is you don't have the same tuning options available for Windows 2000 as you do for UNIX.

Optimizing FTP Downloads

If your site offers a large volume of data for FTP downloads, you will obtain better performance for these downloads, and for other server accesses, by hosting the downloadable data on a separate server. This releases resources on your *iAS* servers for handling HTTP and application requests.

Setting Swap Space and Distributing the Load

In setting up an *iAS* site, make sure swap space use doesn't exceed 75 percent to 80 percent. *iAS* generates new processes over time. You should set up each *iAS* machine with three times as much swap space as physical memory. If your system begins swapping or paging excessively, you might be running too many processes on your system. In this case, add more memory or additional nodes (machines) to your site to distribute the load. Another possible cause of excessive swapping and paging is memory leakage in applications.

Summary

iAS is a complex and powerful product. With a multitude of configuration and tuning options available, *iAS* is the perfect application server for any business, from a small company to the largest dot-com in the world. Invest the time to understand the Apache Web server completely and, in turn, *iAS* inside and out—and it will pay off ten times over. If you simply read the quick tips in the beginning of this chapter, and then read the summary, I highly recommend you find the time to read and understand this chapter. I hope the “Getting Started” section provided you with a basis from which to start configuring *iAS*.