

I INSTALLED 9IAS AND PORTAL--NOW WHAT?

Dan Norris, norris@celeritas.com, Celeritas Technologies, LLC

INTRODUCTION AND OVERVIEW

Oracle 9iAS is Oracle's application server platform. It is made up of many smaller components that integrate to provide all the services you could want from an application server product. Companies may use 9iAS for any application server need, Java or PL/SQL. Because of the extensible nature of the server architecture, new functionality will not require a re-architecture of the environment.

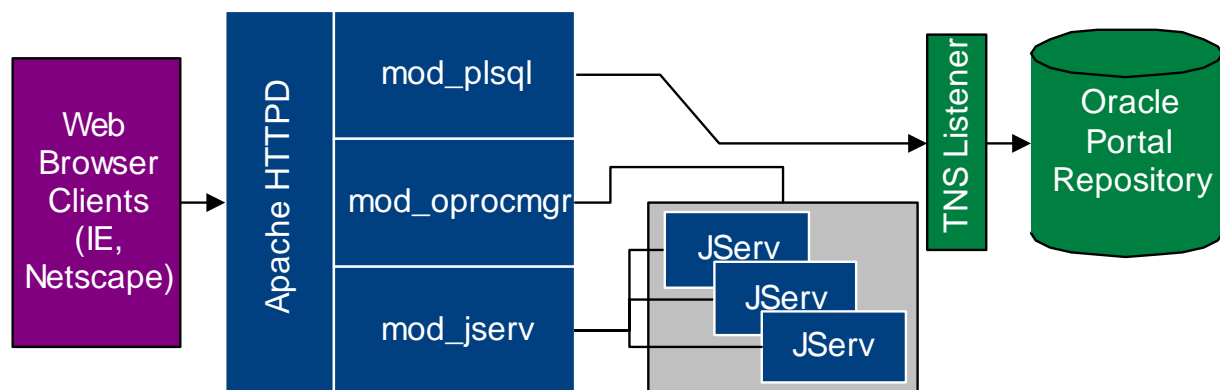
Oracle Portal is a web-based application used to build and deploy portals. A portal usually offers personalization and a way to add and remove parts of the site you see in a modular way. You can also modify the location of certain information on a given page. For example, you may put information you use infrequently at the bottom of the page and frequently used information, like a phone book search tool, at the top of the page. The Oracle Portal application offers a framework for your company to develop things called portlets that can be added to the list of choices users will build their pages from.

The Oracle 9iAS and Oracle Portal products are not hard to install properly, but finishing their configuration it is not always clear. And, if configured properly, the Oracle Portal product can be scalable, extensible, secure, and easy to maintain. Unfortunately, it is difficult to find the information needed to accomplish all those goals in one place.

This paper introduces the 9iAS and portal architecture and documents how to configure some parts of the environment to achieve the functionality and manageability necessary to make a successful implementation. The information included here comes from practical installation and configuration experiences, Oracle documentation, white papers, and various Internet sites and mailing lists. While all of the tips and tricks documented here worked for the customers I've assisted, they may not work for you. In some cases, they may work but leave you in an unsupported configuration. Either way, I am not guaranteeing any amount of success should you decide to implement these suggestions. Also, my experience is mostly on UNIX systems, so all of the pathnames and experiences below were drawn from working with UNIX systems. If you're working on NT systems, your mileage may vary.

9IAS ARCHITECTURE

The 9iAS architecture consists of many components, but only a subset of them are used by most installations. A discussion of components related to Oracle Portal is the focus of this paper. This section gives an overview of the most frequently used components and their purpose in the 9iAS environment.



APACHE HTTP SERVER

The core component of the 9iAS product is the Oracle HTTP Server powered by Apache. The Apache HTTP Server is widely used on the internet and powers more web sites than any other HTTP server software. It is maintained by The Apache Group and is freely available to anyone that cares to download it (including the source code for the entire server). Oracle has made some adjustments to the Apache server available from The Apache Group and redistributes that modified version as the heart of the 9iAS platform.

A key feature of the Apache server is its extensibility. There is a well-documented API that can be used to “plug in” custom modules at an object level. These modules allow the Apache server to be integrated directly with other packages while still maintaining high efficiency. Three of these modules (`mod_jserv`, `mod_oprocmgr` and `mod_plsql`) are covered in this paper.

| <u>Configuration Files</u> | <u>Purpose</u> |
|---|---|
| <code>\$IAS_HOME/Apache/Apache/conf/httpd.conf</code> | Main Apache HTTP server config file |
| <code>\$IAS_HOME/Apache/Apache/conf/oracle_apache.conf</code> | Includes many other configuration files |
| <code>\$IAS_HOME/Apache/Jserv/etc/jserv.conf</code> | Configuration parameters for <code>mod_jserv</code> |

| <u>Log Files</u> | <u>Purpose</u> |
|---|------------------------|
| <code>\$IAS_HOME/Apache/Apache/logs/access_log</code> | HTTP server access log |
| <code>\$IAS_HOME/Apache/Apache/logs/error_log</code> | HTTP server error log |

Tips for the Apache HTTP server configuration:

- Make sure the `ServerName` is set to exactly the same name used when running the `ssodatan` script (if you have to run it—it won't hurt to run it multiple times).
- Make sure that you've examined each `.conf` file included in the main `httpd.conf` (and any files that those include as well) to make sure you know what ports your HTTP server is listening on. Pay particular attention to the `oem.conf` file as most sites don't make use of the OEM installed with 9iAS, yet it opens up port 3339 for incoming HTTP traffic unnecessarily.
- Consider using the `rotatlogs` program that comes with every Apache installation. Its use is documented well in the Apache documentation. It will help ensure that your access logs don't grow uncontrollably.
- If your site develops any custom portlet providers, you may need to add a symbolic link from `$IAS_HOME/Apache/Apache/htdocs/virtualdir` to the directory where the portlets are located. My experiences have been that the parallel servlet hard-codes the `htdocs` directory as the only place where web content resides. With operating systems where symbolic links are not available (such as Windows NT), you may have to actually put the content in the `htdocs` directory.
- Use the Expires HTTP headers to help cut traffic. To do this, you'll need to become familiar with the `ExpiresActive` and `ExpiresByType` directives for the Apache HTTP server (put them in the `httpd.conf` file). I found it useful to set these headers for the image types (JPG, GIF, etc.). The mime types for those image types can be found in the `$IAS_HOME/Apache/Apache/conf/mime.types` file. Here's a sample to tell clients to cache JPG and GIF files for 8 hours from the time they first access them:

```
ExpiresActive On
ExpiresByType image/gif A28800
ExpiresByType image/jpeg A28800
```

When these are used, the HTTP server will tell clients when they first access a GIF or JPG file that they should not request that file again for 8 hours. This dramatically cuts down on the volume of requests your HTTP server has to handle.

- If you are using SSL for the entire Portal (not just Login Server), read Metalink NOTEs 136153.1 and 161099.1. In addition to reading the instructions there, I also needed to modify the following line from my httpd.conf file:


```
SetEnvIf User-Agent ".*MSIE.*" nokeepalive ssl-unclean-shutdown
```

 to read:


```
SetEnvIf User-Agent ".*MSIE.*)$" nokeepalive ssl-unclean-shutdown
```

 This directive changes the way that SSL connections behave when they come from an IE browser. Because of the way that the page servlet works, we need to distinguish between “real” IE clients and recursive requests that are generated as a result of an IE client requesting a portal page. This slight modification corrects that problem.
- For troubleshooting, it is helpful to enable the server status page by uncommenting the <Location /server-status> section of the httpd.conf file and restarting the HTTP server. This will enable you to view server status information by going to http://yoursite.com/server-status. You may also wish to turn off extended status information, which can severely impact performance if the server status location is enabled. That is accomplished by setting the ExtendedStatus directive to Off in the httpd.conf file. Oracle9iAS ships with the ExtendedStatus turned on by default.
- If you’re curious about the configuration, you may also display server configuration information by enabling the server-info screen. Uncommenting the <Location /server-info> section of the httpd.conf file will enable this feature. Once you’ve enabled it and restarted HTTPD, go to http://yoursite.com/server-info to view server configuration information. This information is helpful if you’re trying to make sure that a configuration directive isn’t being overridden somewhere else or checking to make sure the right environment variables are set for the HTTP server.

APACHE JSERV, MOD_JSERV AND MOD_OPROCNGR

JServ is another Apache Group piece of software that is capable of running Java Servlets. In the 9iAS environment, it is mainly used by Oracle Portal for running the parallel page servlet and custom portlet provider servlets. The JServ engine is a standalone background process that receives requests from the Apache HTTP server via the mod_jserv module and is managed by the mod_oprocmgr module.

The mod_jserv module enables the Apache server to communicate with the Apache JServ servlet engine. It is configured with several directives in the httpd.conf file of the HTTP server. On a default installation, these directives are put in a separate file, jserv.conf, found in \$IAS_HOME/Apache/JServ/etc and an include directive is placed in httpd.conf to include the jserv.conf file. The parameters in jserv.conf configure the communication, logging, and virtual directory mappings for the mod_jserv module.

The mod_oprocmgr module is available in 9iAS versions 1.0.2.2 and higher. It was written by Oracle to address the management of the JServ process. The mod_oprocmgr module makes it possible to start only the HTTP server and have one or more JServ processes started and managed for you. Prior to this module, one JServ process could be started and managed by the HTTP server, but most sites require multiple JServ processes and were forced to manage the JServ processes manually. From an administrator's point of view, the mod_oprocmgr module is one of the best features Oracle added to this product.

Tips on JServ, mod_jserv, and mod_oprocmgr:

- Run the same number of JServs as you have CPUs on the system. Some documentation suggests that running JServs equal to twice the number of CPUs is recommended, but that was not the case in the sites I’ve worked with.
- Use mod_oprocmgr if at all possible. If it is not included in your release, strongly consider upgrading to version 1.0.2.2 or higher and take advantage of this module.
- In the \$IAS_HOME/Apache/Jserv/etc/zone.properties file for the JServ configuration, note that the JSP servlet runs from this zone. You may wish to add some initArgs to this servlet like debug_mode, developer_mode, and alias_translation. Refer to the OracleJSP developer’s guide and reference for more information about those parameters. Specifically, setting debug_mode and developer_mode to false (true by default) were significant performance boosts in my experiences.
- Set up a separate servlet zone for each application that needs one. A servlet zone is a collection of servlet repositories that share a common context. By creating multiple zones, you can isolate servlets from one another so that if any one of them has a problem, it cannot affect servlets in other zones. I have gotten in the habit of setting up separate servlet zones for each portlet provider as well. By default, there is only one servlet zone and its configuration file is \$IAS_HOME/Apache/Jserv/etc/zone.properties. To create new zones, you will have to create a properties file

for the zone (start with a copy of the zone.properties), modify the zones parameter in the jserv.properties file, and restart JServ.

- In each servlet zone on your production server, turn off auto reloading for classes and files. You will find parameters for disabling the auto reloading features in each servlet zone's properties file.
- Check min (-ms) and max (-mx) memory heap sizes for the JServs in the startup or wrapper.args parameter in the jserv.properties file. Make sure they are set to the same value and that they are sufficiently large (no out of memory errors in the jserv log files).
- For troubleshooting, it may be helpful to enable the JServ status handler. Uncommenting the `<Location /jserv/>` section of the jserv.conf file and restarting the HTTP server enables it. Then, just go to `http://yoursite.com/jserv/` (note the trailing slash) and you can view information about your JServ configuration and each running JServ. Note that if you do not secure access to this page in some way, others may be able to stop your JServs through this interface. Therefore, I would recommend only enabling it when it is required and disable it when you're finished troubleshooting.

| Configuration Files | Purpose |
|---|--|
| <code>\$IAS_HOME/Apache/Jserv/etc/jserv.properties</code> | Main JServ configuration file |
| <code>\$IAS_HOME/Apache/Jserv/etc/zone.properties</code> | Root servlet zone configuration file |
| <code>\$IAS_HOME/Apache/Jserv/etc/jserv.conf</code> | Configuration parameters for mod_jserv |

| Log Files | Purpose |
|--|--|
| <code>\$IAS_HOME/Apache/Jserv/logs/jserv.log</code> | JServ log file |
| <code>\$IAS_HOME/Apache/Apache/logs/error_log</code> | mod_jserv logs go into HTTPD error log |

MOD_PLSQL

The other module heavily used with Oracle Portal is the mod_plsql module. If you are familiar with the Oracle Web Application Server product, the functionality it provided with the PL/SQL cartridge is provided in 9iAS with the mod_plsql module. If you have never heard of either one, don't feel bad—you're not alone. The whole purpose of this module is to allow PL/SQL packages and procedures to become web-based applications. The mod_plsql module provides a gateway between the web server and the database(s) that are housing application(s) written in PL/SQL.

The PL/SQL applications use special API calls from a set of packages referred to as the OWA (Oracle Web Access) packages. These packages must exist in any database that wishes to house a PL/SQL application. These packages come with the 9iAS installation and the version used must be compatible with the mod_plsql version used.

This module is also capable of caching results of certain procedures according to its configuration. Caching will generally increase scalability of an application, but careful evaluation is always necessary to identify how much and what type of caching will benefit an application.

The Oracle Portal product is mostly comprised of PL/SQL packages and database tables, so the mod_plsql module is one major enabling technology behind Oracle Portal.

Tips on mod_plsql:

- On UNIX systems where Apache HTTPD uses a process model for scalability, database connections are opened for each process spawned. For Oracle Portal implementations, as many as 3 database sessions can be opened for each of these processes. The httpd.conf parameter MaxClients governs how many processes can be running at any given time. If the number of users attempting to access the site grows above the number set by MaxClients, those requests are held in a TCP queue, not dropped. You should determine whether your database can handle 3 x MaxClients by checking the database's processes parameter.
- In all versions of the Portal I've worked with, you must go to the PL/SQL gateway configuration page and at least view the configuration of the DADs in order for the DAD password to be encrypted in the configuration file (`$IAS_HOME/Apache/modplsql/cfg/wdbsvr.app`). In some older versions, the passwords were never encrypted. When going to this configuration page, make sure you are forced to authenticate. If you are not (pre-1.0.2.2 9iAS versions

did not require you to authenticate), configure basic HTTP authentication for the gateway configuration URL by using the AuthType and require directives in the Apache configuration files (I put it in \$IAS_HOME/Apache/modplsql/cfg/plsql.conf). See the Apache docs and NOTE 108661.1 on Metalink for more information.

- Monitor the mod_plsql cache directory (\$IAS_HOME/Apache/modplsql/cache) to ensure that it is maintaining itself appropriately. In at least one instance, a bug in mod_plsql caused the cache directory cleanup to fail, resulting in a disk drive filling up. If the cache directory fails to maintain the size specified on the Listener Gateway Settings page (or close to it), I found that removing files from the cache did not harm functionality for anyone. It appeared in my testing that the removed file is recreated if necessary on the next access.

| <u>Configuration Files</u> | <u>Purpose</u> |
|---|------------------------------|
| \$IAS_HOME/Apache/modplsql/cfg/plsql.conf | mod_plsql configuration file |
| \$IAS_HOME/Apache/modplsql/cfg/wdbsvr.app | DAD definition file |

| <u>Log Files</u> | <u>Purpose</u> |
|---|--|
| \$IAS_HOME/Apache/Apache/logs/error_log | mod_plsql errors go into HTTP server error log |

OTHER MODULES INCLUDED FROM APACHE GROUP

Many of the modules that Apache delivers as part of their software are also included in the default Oracle 9iAS installation. Some of the most popular modules are: mod_perl, mod_cgi, mod_rewrite, mod_setenvif, mod_status, mod_expire, mod_speling, and mod_ssl.

ORACLE PORTAL ARCHITECTURE

The Oracle Portal architecture consists of the portal repository (a database schema holding many tables, indexes, procedures, and packages), the parallel page servlet, and portlets.

PORTAL REPOSITORY

The portal repository is where all portal objects are stored. It is recommended to create a specific Oracle instance for the portal repository. To create the schema objects, the Oracle Portal Configuration Assistant (OPCA) is run from the 9iAS installation. In just a few steps, the wizard collects some information about the destination where you chose to create the repository and then creates all the objects. This part is pretty simple. Once it finishes, you should have a valid portal repository. One thing to keep in mind is that the assistant you run determines the Oracle Portal version you'll be creating. This seems obvious, but if you install multiple versions of 9iAS on the same system, take extra care to run the desired version of the OPCA.

In one version 3.0.8 portal repository, the PORTAL30 schema owned 343 tables, 495 indexes, 608 packages, 10 procedures, 7 functions, 153 views, 72 types, 70 triggers, 65 sequences, 38 java classes, and 37 LOBs. In that same repository, the PORTAL30_SSO (login server) schema owned 75 tables, 142 indexes, 184 packages, 7 procedures, 2 functions, 44 views, 41 types, 27 triggers, 39 sequences, 2 java classes, and 4 LOBs. With all those objects to create (the one assistant creates them all), it's no wonder the assistant runs as long as an hour or two depending on the configuration and systems involved.

Tips for creating the portal repository:

- Make sure you have installed the Java and interMedia options before you create the repository. During creation, these subsystems are detected and certain packages are installed based on the status of them. If you don't have the Java option installed, the OPCA will refuse to create the repository.
- Create separate tablespaces for logs, docs, and default portal objects. Then, when the OPCA prompts you, choose the appropriate tablespace names from the drop-down lists.
- Make sure the database you create the repository in has the latest patches installed. Installing patches later is more difficult due to the presence of the Java and interMedia options.

PARALLELSERVLET (A.K.A. PAGE SERVLET)

The parallel servlet is what does most of the work when assembling a portal page to present to the end user. It runs in the JServ(s) that are part of the 9iAS environment. When an HTTP request for a portal page arrives, the parallel servlet breaks up the request into multiple separate HTTP requests, one for each portlet on the requested page. It assigns the recursive requests to other threads in itself to be retrieved in parallel. Once all threads have returned from their request (could be an error condition or successful content), the master thread assembles the page and presents it to the user. The parallel servlet is capable of running in HTTPS-only environments as well, but there is some additional setup required for both 9iAS and the parallel servlet.

CONFIGURATION PARAMETERS

The configuration parameters for the parallel servlet are shown below. To set these, you should add these parameters as `initArgs` lines to the `zone.properties` file for the root servlet zone (`$IAS_HOME/Apache/Jserv/etc/zone.properties` by default). These would look something like this:

```
servlet.page.initArgs=stall=100,poolSize=50
```

The defaults for the parameters below are not documented clearly in any portal release until 3.0.9. In the portal 3.0.9 documentation there is a table of defaults listed (for version 1.0.2.2, it can be found at http://technet.oracle.com/docs/products/ias/doc_library/1022doc_otn/portals.102/a90096/cgmidfw.htm#1001621), but I have determined that at least a couple of the listed defaults are inaccurate, so declare the parameters if you want to be sure. Here are the parameters available for 9iAS 1.0.2.2.

| Parameter Name | Description/Default Value |
|----------------|--|
| logpath | path for log file, default is JServ log file |
| logmode | setting to debug will enable debug outputs, default is null |
| showError | enables display of error messages in Oracle Portal user interface, default is TRUE |
| poolSize | number of fetchers to use, default is 25 |
| stall | time in seconds to wait for connection establishment, default is 120 |
| requesttime | default time in seconds to wait for content, default is 40 |
| httpsports | SSL ports seperated by colon, no default |
| prefix | default modplsql prefix, default is /pls |
| offlinePath | allows you to take Poral offline for maintenance; all requests to parallel servlet are sent contents of the file specified by this parameter |
| proxyHost | Defines the host to use for requests which need to go through a proxy server, no default |
| proxyPort | Specifies the port to use for requests which need to go through a proxy server, no default |
| proxyIgnore | Specifies those domains you want ignored for the proxy settings. According to the HTTP 1.1 standard, domains are required to start with a ".". For example, .oracle.com is valid, oracle.com is not. No default. |
| useScheme | Forces the Parallel Page Engine to use the protocol specified on this line, valid values are http or https, default is http |
| usePort | Forces the Parallel Page Engine to always use the specified port number for all requests |
| cacheBuffer | Specifies the size of the memory buffer to use to return a cached page from the middle tier. For maximum efficiency, set this value as close as possible to the actual byte size of a complete page. If the value is set too small multiple reads are performed to the disk. The default is 32768 bytes. |

If you have installed the 3.0.9.8.2 portal patch, you also have the following parameters available:

| Parameter Name | Description/Default Value |
|----------------|--|
| minStall | The minimum stall time (in seconds) allowed for a contentfetcher before it assumes there is a problem and times out. This value defaults to 120. |

| | |
|------------|---|
| minTimeout | The minimum timeout (in seconds) that may be sent by a portlet. The default is 5. Note that this is different than the requesttime parameter which sets the timeout for portlets that do not send one. minTimeout sets a minimum for all portlets regardless of how individual portlets are configured. |
|------------|---|

TUNING

The parameters above can help tune your portal installation significantly. My experiences are that tuning the custom portlet code (if any), running as many JServ processes as you have CPUs, and setting the requesttime, stall, minStall, and minTimeout parameters appropriately made the biggest difference.

Tuning custom portlets can be as simple as removing unnecessary nested tables from their HTML code. Depending on the portlet's purpose, it may be necessary to tune logic in the portlet just as you would for any other application. The skills required to tune that logic will depend on what type of provider provides that portlet. If it is a database provider, then the logic will be in PL/SQL in a database. If it is a web provider, the logic could be in nearly anything Java (JSP, Servlets, EJBs, etc.). Netscape browsers have an especially difficult time rendering large numbers of nested tables efficiently, so you can make some simple changes to resolve that issue if your portlets add more nested tables to your portal pages.

The biggest difference will likely be found with tuning the number of JServs running. By default, Oracle9iAS ships with one JServ configured. With more than one JServ running, you need to configure mod_jserv to balance requests across all of them equally. I've not worked with any customers that are running JServs on a different host than the Apache installation, so making sure you're using the shared resources efficiently is critical to the performance and scalability of the portal site running on a single system. With running the same number of JServs as we had CPUs on the system, we saw the best performance-to-scalability ratio. When running less JServs, we would frequently see portlet timeouts and requests would hang even though the HTTPD had lots of headroom. When running more JServs than CPUs, we saw a terrible CPU bottleneck and the run queue length (as reported by vmstat and uptime) would spike so high the system was nearly unusable until you stopped the JServs.

The requesttime parameter is supposed to limit the amount of time a parallel servlet thread waits for a response before it gives up and returns an error. This parameter is used only if the portlet does not have a timeout listed for it already. For the portlets delivered with the portal, it is unclear whether or not they have the timeout set since there is no provider.xml file for them. For custom portlets, you can find the timeout for a portlet by consulting the provider's configuration file (provider.xml) found in the provider_root directory.

Tips for the Parallel Servlet:

- Make sure you have installed the latest portal patches (available via Metalink -> Patches). For the 3.0.9 portal release, one of the patches updates the parallel servlet code and makes available a couple more initArgs settings that you will likely want to use with your site.
- Since the Parallel Servlet runs in the root servlet zone, try to limit the number of other servlets running in that zone. You could even relocate the OracleJSP servlet to another zone if you wish, but it is unlikely that you would see much benefit from moving just that one unless you use JSP heavily on your site.

PORTLETS

The portal pages you design will be comprised of individual portlets. I think of individual portlets as "sub-pages" that together comprise a whole web page. Oracle Portal provides the framework for these portlets to be displayed as one page and also allows users to create pages, add portlets to their pages, customize the portlets on a page, and reposition their portlets within a given page.

PORTLET PROVIDERS

Oracle provides portal customers with a number of portlets right out of the box. However, if you're going to be using the portal as the main source of information at your company, you (or someone who pays your salary) will likely want to create custom portlets to display things like the current local weather, company stock quotes, cafeteria menu, address book lookup, or other specialized bits of information. In order to get your own portlets on a page, you'll have to create the portlets and a provider for them as well. The Portal Development Kit (PDK), downloadable from <http://portalstudio.oracle.com/>, will need to be installed on your Portal to provide the APIs developers need to create a portlet provider. On <http://portalstudio.oracle.com/>, you can also find lots of code snippets to get you started writing your own portlets.

There are two types of providers to choose from. The type you'll use will depend on how you wish to implement your portlets.

DATABASE PROVIDERS

If you want to write your portlets in PL/SQL packages or procedures, you'll be using a database provider. To access your portlets, you'll be making heavy use of the `mod_plsql` module. Programming your portlets will require you to be familiar with the OWA packages Oracle provides so that you can produce HTML and you will also have to use the PDK.

WEB PROVIDERS

If PL/SQL isn't your bag, no worries. You can write your portlets in nearly any Java technology you wish. The PDK is available with Java APIs as well (it is called the JPDK), so you write your code to do whatever it is you wish and then integrate with the JPDK APIs and you're all set. Additionally, since all the accesses to the web provider are done through HTTP, you can run the portlets on any servlet (v2.0+) engine you wish if the JPDK has been installed on that servlet engine.

HOW A PORTAL PAGE IS ASSEMBLED

Oracle has an excellent white paper (Page Generation and Assembly Scalability in Oracle9iAS Portal) available from <http://technet.oracle.com/> that details the page assembly very well. For the purposes of this paper, I'll outline the process with the highlights and leave the details up to those that wish to read the white paper from Oracle.

As always, nothing happens at a website without a customer, so it all starts with the user request for a portal page (for example, the homepage). Once the site receives the request (<http://yoursite.com/pls/portal30/portal30.home>), it runs the PL/SQL procedure named HOME in the portal30 schema of the database configured as part of the DAD named portal30. That procedure will return a redirection to the client. That redirection will send the client to some URL that looks like <http://yoursite.com/servlet/page?pageid=1,11&schema=portal30...> The important thing to note is that the page servlet has now entered the picture. From here on out, the user will be accessing the page servlet (a.k.a. the parallel servlet).

After the page servlet receives the request for the portal home page, it retrieves that page's metadata from the database by making an HTTP request to <http://yoursite.com/pls/portal30/...> and determines that there are 5 portlets (for example) on the portal home page. It then dispatches 5 worker threads from its thread pool (the `poolSize` parameter) to build the 5 portlets. Once all 5 workers are finished, the coordinator that received the initial request from the client assembles the 5 individual portlets into a page using the metadata retrieved earlier and returns the page to the client.

From the perspective of the HTTP server, it took at least 7 requests to satisfy the end user (1 initial request that resulted in a redirect, 1 request to the coordinator page servlet, and 5 requests by the worker threads—assuming that the provider was also on this HTTP server). In the HTTP server's access logs, it is useful to note how many requests were from clients that are not the local host. You can use simple `grep` commands on UNIX to remove those recursive requests to be left with a log that contains only requests from "real" clients. You may wish to prune that log further because to bring up the portal home page, our example client actually made two requests: one initial request and a second as the result of a redirect from the first request. Overall, it is not straightforward to determine the number of actual client requests served by a portal site since there are many redirects (resulting in what I call redundant requests) and a very large number of recursive requests initiated by the page servlet.

Another consideration is that the network traffic generated on the portal server is actually more than double that of a "normal" website that doesn't use a recursive method to build the pages. The reason for this doubling effect is that each recursive request returns the portlet data plus some overhead and then once all recursive requests are filled, the coordinator thread repackages that same data and sends it back to the client. So, the HTTP server sends that data out once to fill the recursive requests and then a second time to satisfy the actual client request.

The recursive nature of the site can also complicate internet or extranet deployment of the portal environment. Many sites wish to have the database server on their internal network for safety and maintainability, but need to have the HTTP server in a DMZ or outside their firewall. A clear understanding of how the recursive requests are initiated and when and how the database is accessed will be absolutely necessary to deploy and maintain an Oracle Portal site on the internet.

INTERMEDIA

INTERMEDIA DEFINED

interMedia is an Oracle database subsystem that can be used to index many different types of files stored inside or outside of the database including text, html, pdf, video, audio, and binary files. As of Oracle8i, interMedia is included with the Oracle8i Enterprise Edition license at no extra charge. In previous releases, this option, referred to as ConText, was an extra-cost option and only provided a subset of the capabilities it does currently.

HOW PORTAL USES INTERMEDIA AND HOW INTERMEDIA WORKS

The portal uses interMedia to enable searches within file upload areas, called content areas, but it also can enhance the search features of the rest of the site as well. The files uploaded into portal can be literally anything. What makes interMedia superior is that it will determine the file type and then apply appropriate filters to extract any indexable pieces of the file without indexing binary data. The interMedia processor figures all that out without ever telling it what the file's type actually is. There is a list of supported file types in the interMedia administrator's guide in the RDBMS documentation.

The technicalities of how interMedia runs are pretty interesting. First, to use interMedia, you must create interMedia indexes. This is confusing since an interMedia index isn't really an index at all, but rather a set of objects (4 tables, 2 indexes, and a LOB index depending on the interMedia index type) for the one interMedia index. Furthermore, this index, as it is called, is not kept in sync with the data it is indexing. You must either synchronize it manually or configure an automatic process to perform the synchronization for you.

SETUP AND CONFIGURATION TASKS

Configuring interMedia was one of the most difficult parts of setting up the portal site. I have not known anyone that went through any training on interMedia (including myself), so I'm sure that's part of the reason it was so difficult. I think that you'll benefit from a few important lessons I learned the hard way.

The documentation I found basically said that you had to set the appropriate LD_LIBRARY_PATH environment variable and everything would work. This may have been true, but I couldn't find any documentation to tell me where I needed to set this variable. With multiple ORACLE_HOMEs on most systems these days, it was not practical for us to set the LD_LIBRARY_PATH globally in the login profile for our oracle user, so we had to find a better solution. One resolution to this problem is to replace the file \$ORACLE_HOME/ctx/bin/ctxhx with a short shell script that sets the appropriate environment variables and then executes the "real" ctxhx binary. This approach presents a couple of potential problems: 1) Oracle Support is probably going to have difficulty supporting it, and 2) Installing patch sets requires some extra care. Those conditions were acceptable to the sites I've worked with, so I renamed \$ORACLE_HOME/ctx/bin/ctxhx to \$ORACLE_HOME/ctx/bin/ctxhx.real and then created a shell script named \$ORACLE_HOME/ctx/bin/ctxhx that looked like this:

```
#!/bin/sh
ORACLE_HOME=/u01/app/oracle/product/8.1.7   ### put in the appropriate O_H path here
PATH=$ORACLE_HOME/ctx/bin:$PATH
LD_LIBRARY_PATH=$ORACLE_HOME/ctx/lib:$ORACLE_HOME/lib
export ORACLE_HOME PATH LD_LIBRARY_PATH
$ORACLE_HOME/ctx/bin/ctxhx.real $*
```

It is important to note that the ORACLE_HOME you specify in the script is the RDBMS ORACLE_HOME where the portal repository is installed, not the IAS_HOME directory. That is because interMedia is a feature of the RDBMS that runs in the database and is not a feature of 9iAS. This solution does involve changing a file in the ORACLE_HOME directory, so if you're not comfortable with that, then you will need to find another solution. The only disadvantage of using this method is that anytime patches are added to this ORACLE_HOME, you will need to ensure that your ctxhx script was not overwritten. I find it helpful to make a copy of the ctxhx script and call it something like \$ORACLE_HOME/ctx/bin/ctxhx.script so that a patch installation will not overwrite it. If a patch does overwrite the ctxhx script file, repeat the procedure of moving it to ctxhx.real and recreating the shell script (or copying it from your backup file) so that you will get the benefits of the new patch to the ctxhx file.

I also found a genuine lack of communication or understanding about how to set up a synchronization process among Oracle Support groups. Apparently, changes were made in version 8.1.7 that were not disseminated well into the support organization.

Here's the scoop: Prior to version 8.1.7, a separate background process (not a database background process, but a daemon you had to start outside the database) called ctxsrv handled the synchronization. In version 8.1.7, the ctxsrv method of updating the interMedia indexes was deprecated in favor of using a PL/SQL package named CTXSYS.CTX_DDL. This package could be scheduled in the database's job queue with a short interval to check for updates often. I found it useful to write a short package named CTX_MAINT (under the PORTAL30 schema) to handle the sync methods to ensure that no synchronization was attempted if the index was still being created or became invalid for any reason. A package listing is in the next section. In

trying to find the right answer about whether the `ctxsrv` or `CTX_DDL` was the correct, supported method to synchronize the interMedia indexes, I got in the middle of a dispute between the interMedia support group and portal support group at Oracle. In my opinion, which method to use for synchronization has nothing to do with the Oracle Portal (since interMedia can be used independently), so I followed the advice from the interMedia group and everything worked fine. Plus, I didn't have to figure out how to manage another background process and the username/password it requires to connect to the database!

CTX_MAINT PACKAGE LISTING

```
CREATE OR REPLACE PACKAGE ctx_maint
IS
  v_opt_timeout_minutes CONSTANT NUMBER := 30;
  PROCEDURE sync_indexes;
  PROCEDURE optimize_indexes;
end ctx_maint;
/
CREATE OR REPLACE PACKAGE BODY ctx_maint
IS

  PROCEDURE sync_indexes IS
  BEGIN
    FOR i IN (SELECT idx_name FROM ctx_user_indexes
              WHERE idx_status = 'INDEXED'
              ) LOOP
      CTX_DDL.SYNC_INDEX(i.idx_name);
    END LOOP;
  END sync_indexes;

  PROCEDURE optimize_indexes IS
  BEGIN
    FOR i IN (SELECT idx_name FROM ctx_user_indexes
              WHERE idx_status = 'INDEXED'
              ) LOOP
      CTX_DDL.OPTIMIZE_INDEX(i.idx_name
                             ,CTX_DDL.OPTLEVEL_FULL
                             ,v_opt_timeout_minutes);
    END LOOP;
  END optimize_indexes;

end ctx_maint;
/
```

Tips for interMedia:

- Use the `CTX_DDL` package for synchronization.
- Make life simple by replacing the `ctxhx` binary with a script as outlined above.
- Check the `CTX_INDEX_ERRORS` view regularly to find errors as they occur. Also check the `CTX_INDEXES` view to ensure that each index has status `INDEXED`. Also check the background dump destination for trace files related to interMedia.
- If you do not need interMedia, don't use it.

HOW LOGIN SERVER FITS IN

The Login Server, if you are unfamiliar with it, is a product designed to provide a single sign-on for web-based applications. It comprises many tables, indexes, and PL/SQL packages that reside in an Oracle8i or higher database. These components are accessed by users via the Oracle HTTP Server and `mod_plsql` module.

I think one of the most difficult parts of 9iAS to understand is the Login Server. Some documentation suggests that the Login Server is a part of the portal while other documentation says it is an independent component. What probably confuses most people is that the Login Server is installed/created with the Oracle Portal Configuration Assistant (OPCA), leading you to believe that it is a part of the portal. However, it is possible to use Login Server independently. In fact, multiple portal sites can use the same Login Server to authenticate its users.

The Login Server identifies two different types of applications that it will authenticate users to. First, partner applications are those that have been integrated with the Login Server by writing to the Single Sign-on SDK APIs. Those APIs are available in PL/SQL and Java languages.

The other way to utilize Login Server is to configure external applications. External applications are those that have not implemented the Login Server APIs from the Single Sign-on Software Development Kit (SSO SDK) and have no idea that a user is logging on to them with a Login Server. This works by storing your login credentials in the Login Server and when you visit a site that requires login, the Login Server intercepts the login page and submits your stored username and password for you. The end user experience is the same as if they had logged on to the application directly except that they were not prompted for the login. This means that once a user authenticates to the Login Server once (which happens the first time they attempt to access a protected application), they are automatically signed on to all partner and external applications and receive no further authentication prompts.

The only time the user is re-authenticated is if they close all browser windows (since Login Server uses a cookie that is removed when the browsers are closed), if they specifically log out of the Login Server, or if the cookie in their browser has passed a timeout limit configured by the Login Server administrator. This protects the user if they forget to close all browsers or if they leave their terminal unlocked for an extended period of time.

One final general note about the Login Server—it only handles authentication. Authentication and authorization are two very different things. Authentication is identifying a user, usually by a username/password combination, but it could be just about anything: fingerprint, retina scan, voice pattern recognition, or PKI credentials to name a few. Authorization is ensuring that the identified user has the privilege to perform some action. Of course, authorization always comes after authentication since you cannot authorize a user that you have not yet identified.

In the Login Server architecture, the application is still responsible for authorization and privilege management. That usually means that even though the application doesn't store passwords, someone will still need to manage users in the application for the purposes of individual privilege management.

PORTAL IS JUST ANOTHER PARTNER APPLICATION

While the integration between the Portal and Login Server is automatically done at the time they are installed (if installing them into the same database), the portal is just another partner application and receives no special treatment by the Login Server. Other partner applications can be integrated with the same Login Server without difficulty and receive the same services as the Portal application does.

If you wish to install the portal, but use a Login Server located on another system, it is simple to make the switch in the Login Server and Portal such that it is registered with the new Login Server.

SINGLE SIGN-ON SOFTWARE DEVELOPMENT KIT (SSO SDK) FOR YOUR OWN PARTNER APPLICATIONS

To write your own partner application, you will have to retrieve the Single Sign-on SDK from Oracle. This has only recently become a part of the Portal Development Kit (PDK) available from <http://portalstudio.oracle.com/>. I had great difficulty in locating the SSO SDK when I first attempted. In fact, the people I spoke to at Oracle Support also did not know what I was asking for and had to ask development to send the SSO SDK. They have since told me that the SSO SDK is delivered with the PDK now, so that should get you over the first hurdle.

If writing a PL/SQL-OWA application, you will likely find it straightforward to make the appropriate PL/SQL Login Server API calls from your partner application. To use the Java APIs, there is a significant amount of setup work to be done in the database to support those APIs. It is all well documented in the SSO SDK instructions and release notes, so you shouldn't have any trouble. The interesting part about using the Java APIs is that you still have to use the PL/SQL APIs too. That is because some of the exchanges between Login Server and your Java application are encrypted. The encryption and decryption routines are written in PL/SQL and only reside in the database, so your application will have to connect to the database in order to run them. That means you'll have a hard-coded username and password in your Java application that will be critical to getting this environment working.

If you have a choice, it may be easier to integrate your partner application if it is written in PL/SQL. However, it is still possible to accomplish it either way.

LOGIN SERVER CONFIGURATION

There are currently (in 9iAS 1.0.x) two options for the Login Server repository. The main reason that applications should integrate with the Login Server is to externalize the authentication of the application's users. Basically, this means that the application will not have to store passwords, but it also means that if all applications use the Login Server, passwords will only be stored in one place. Most would agree that the fewer places passwords are stored, the less chance they have of being compromised.

INTERNAL (TABLE)

The internal repository is just a table stored in the Login Server schema in the database. The passwords are encrypted before they are stored, so retrieving the contents of the table does not gain you access to all account information.

The internal repository is the default and is the easiest to implement, but may not be the easiest to manage. Bulk changes (account additions, deletions, et cetera) can be made via PL/SQL packages in the Login Server schema. One confusing note about the PL/SQL packages you use to manage accounts: there is a number of packages in the Login Server schema that are also in the Portal schema. To manage the Login Server accounts, you should use the packages in the Login Server schema. If you want to manage Portal accounts, use the packages in the Portal schema.

EXTERNAL (LDAP)

INSERT DIAGRAM OF EXTPROC LISTENER, DATABASE, LDAP DIRECTORY HERE

The alternative to using a table to store usernames and encrypted passwords is to use an LDAP directory. The advantage of using an LDAP directory is that you may be able to centralize authentication into a single source for your entire enterprise. The LDAP directory used by the Login Server can (and probably should) be used by any number of other applications for just about any other purpose you can imagine. If your LDAP server is not used by anything other than Login Server (or you don't have plans to implement other LDAP-enabled applications), you should probably leave Login Server configured as the default using the internal repository.

The disadvantages of using the LDAP repository for Login Server is that it is complicated to set up and, if your LDAP server is on a separate host from the Login Server database, you introduce a security risk by transmitting the end users' passwords over the network in clear text. Enabling SSL communication between the Login Server database and the LDAP directory is a feature that is supposed to be introduced with 9iAS version 2.0. Once that happens, the security risk will be minimized.

As you can see from the diagram above, the architecture involved with the external repository setup is significantly more complex than the internal repository configuration.

Here are some practical tips on setting up the external repository correctly:

- Make sure you have external procedures set up correctly in the listener.ora and tnsnames.ora files. Additionally, you should add a line like the one below to your listener.ora file in the SID_DESC section for external procedures:
(ENVS = 'LD_LIBRARY_PATH=/usr/ccs/lib:/u01/app/oracle/product/8.1.7/lib:/usr/ucblib')
- Make sure that the KEYs listed in the listener.ora and tnsnames.ora files match exactly. The contents of the KEY attribute are arbitrary, but they must match one another. If configured properly, you should be able to run 'tnsping extproc_connection_data' successfully.
- You will need to configure your LDAP directory (Oracle Internet Directory is the only one supported currently) with at least 3 additional DNs for portal30, portal30_sso, and a dedicated read-only account to use when searching for user DNs. You should also change the default ACLs to deny write and selfwrite privileges for both entries and attributes. If you do not create an LDAP entry for the portal30 user, the portal30 user will be unable to authenticate to the login server. If you create an entry for the portal30 user and give it a password different than the one stored in the internal repository, the password given in the LDAP entry will be the one used. There is no "merging" of the two repositories—you're using exactly one of them at a time.
- As part of the switch to using the external repository, you will need to run the ssoldap.sql script and respond to the prompts for information about the LDAP directory. If this fails, you will need to run the ssolocal.sql script to switch Login Server back to using the internal repository. Switching back and forth does not mean that the two are synchronized in any way.

- Check the path given in the DBA_LIBRARIES view for the auth_ext library. If it is not a fully-qualified pathname (starting with a '/' (slash) on UNIX), then recreate it to have a full path using the CREATE OR REPLACE LIBRARY syntax. After a fresh installation, it is usually not a fully-qualified pathname.

9iAS AND OC4J

Oracle Containers for Java (OC4J) is an Oracle product that can be used with 9iAS, but can also be used separately. It provides many J2EE services to any Java application environment including: servlet 2.2 (and most of servlet 2.3), JSP 1.1, and EJB 1.0 (and significant parts of EJB 2.0). The OC4J is a thinly disguised Orion application server product. Oracle has licensed this product and redistributes it with few modifications under the OC4J name.

I am not a Java programmer, so my focus is to explain how OC4J can be integrated into your 9iAS environment.

DEPLOYING JUST EJBS IN OC4J

If you wish to only use OC4J as a bean container, you don't need to do any special configuration to 9iAS. Any servlet or JSP that wishes to may make callouts via RMI or some such standard communication mechanism to the OC4J container. In this configuration, no special changes are necessary in the JServ or Apache HTTPD configuration files.

DEPLOYING SERVLETS AND JSPs IN OC4J

Using OC4J for servlets and JSPs (as well as EJBS if you wish) is also supported. If you wish to use OC4J for these purposes in a standalone configuration as an application server, you simply need to follow the installation and configuration instructions for the OC4J product.

If you wish to integrate the OC4J servlet and JSP functionality into the 9iAS suite of products, you will have to make some significant changes to the 9iAS HTTPD configuration. You can, however, remove the JServ components from your site all together since OC4J will replace it completely.

USING THE OC4J LISTENER

To use the OC4J listener, you just need to modify your HTML documents to use a different port on your system when running any servlets or JSPs. This will direct the client to access the listener that is part of the OC4J product and they will retrieve their requests directly from the OC4J server.

Most customers will probably choose not to implement OC4J in this fashion for two reasons: 1) The OC4J listener was not intended for use as a general-purpose web server and may not scale as well under heavy loads and 2) The end user experience may suffer as their browser communicates with two different web sites. This also makes it difficult to troubleshoot and is nearly impossible to load balance across all tiers.

USING THE 9iAS LISTENER

If using the 9iAS HTTPD listener, the configuration will be changed to employ the Apache HTTPD's proxy module. While the result is still using the OC4J listener for some traffic, it will only be for servlet and JSP traffic, not the entire site. The main reason that customers will choose this configuration is to have a single point of entry from the end user's point of view and it also enables better logging data since all requests are logged by Apache and analysis of that log shows the entire site's accesses.

USING SOMETHING ELSE?

Unfortunately, these are the only two solutions available at this time. It would be my choice to have a special Apache HTTPD module for the Orion server that could have some special features not available with HTTP. For instance, load balancing or fail over could be managed in an Apache module rather than having to do it with a network appliance of some sort as you would have to do with today's options. Additionally, while HTTP does not incur very much overhead, there may be some advantage to tighter integration between Apache and Orion.

Tips for using OC4J:

- Be sure to read all available documentation before you start. The configuration of the OC4J server is in easy-to-read XML files, but they aren't always easy-to-understand.
- If you will be using OC4J to deploy EJBS, make sure you understand them well enough. OC4J is a good product, but even good products can't make up for poor implementation.

ORACLE FORMS SERVER AND REPORTS SERVER

One short note about Forms and Reports Servers. Everything you need to run these products can be installed with 9iAS if you choose the Enterprise Edition. Running them smoothly and securely is a large topic that could fill another 10-20 pages if covered here. If you have questions or problems running these products, make sure you read all available documentation and search Metalink for current NOTES and patches.

In my experiences, once the configuration files for these products are identified and understood, they are not difficult to manage. Environment variables play a large part of the configuration for Reports Server especially, so take extra care to know when and how to set them correctly.

ORACLE9iAS CACHES

There are two caches delivered as components of the Oracle9iAS product. The first is the database cache, also referred to as the Oracle9iAS cache. This cache is designed to speed up database access and is not supported for use with the portal. In fact, when installing the Oracle9iAS Enterprise Edition, I enter garbage in the sections that prompt for information about the “origin database”. The origin database is the database that you would like to be cached with the database cache.

The other type of caching that Oracle9iAS provides is web caching. This works by setting up a caching web server that your end users will access and if it does not have a document cached, it will access the actual Portal site to retrieve it. Because of the dynamic nature of Oracle Portal, this does not provide much benefit. In my experiences, we were able to cache images and that did help, but the same can be accomplished by using the Expires headers directly from the HTTP server (see Apache HTTPD tips). After some testing, we found that the Oracle Portal did not function properly when using the web cache and a call to Oracle Support confirmed that it is not supported for use with Portal either.

CONCLUSION

Most sites that have implemented 9iAS have struggled to determine where one part of the environment stops and another part begins as is so often done for management purposes. For example, some companies have corporate security departments that manage all passwords and authentication programs centrally, another group to manage the system administration, a database administration group, an intranet group, and an application management group. In most systems, it is possible to find a place to break apart a production environment into parts that each group can manage somewhat independently. With an Oracle Portal implementation, it is very difficult to break up the whole into autonomous chunks that different groups can manage. Many components are required to get and keep a portal site running and once they are bound together, it is nearly impossible to break them apart without affecting neighboring components.

Implementing and managing an Oracle Portal site is something that every company can handle, but maybe not in the same way as they have handled other applications previously. Because of the complexities and tight integration, constant, careful communication is required to ensure good performance and 100% functionality at all times.