Oracle Internet Application Server 8*i*

Migrating from Oracle Application Server

Release 1.0

June, 2000

Part No. A83709-01

ORACLE®

Oracle Internet Application Server Migrating from Oracle Application Server, Release 1.0

Part No. A83709-01

Contents

3 Migrating OAS Cartridges to Apache Modules

4 Migrating EJB, ECO/Java and JCORBA Applications

Index

Send Us Your Comments

Oracle Internet Application Server Release 1.0, Migrating from Oracle Application Server

Part No. A83709-01

Oracle Corporation welcomes your comments and suggestions on the quality and usefulness of this publication. Your input is an important part of the information used for revision.

- Did you find any errors?
- Is the information clearly presented?
- Do you need more information? If so, where?
- Are the examples correct? Do you need more examples?
- What features did you like most about this manual?

If you find any errors or have any other suggestions for improvement, please indicate the chapter, section, and page number (if available). You can send comments to us in the following ways:

- E-mail - iasdocs@us.oracle.com
- Postal service:
  Oracle Corporation
  500 Oracle Parkway, M/S 6op4
  Redwood Shores, CA 94065
  USA

If you would like a reply, please give your name, address, and telephone number below.

If you have problems with the software, please contact your local Oracle Support Services.

Preface

This chapter contains general documentation information about the Oracle Internet Application Server, including conventions used in this manual and contact information at Oracle.

Audience

This document is written for application designers and developers, and others responsible for maintaining the Oracle Application Server website.

Conventions

This manual uses the following typographical conventions:

Convention	Example	Explanation
bold	**tnsnames.ora** **runInstaller** **www.oracle.com**	Identifies file names, utilities, processes, and URLs
italics	*file1*	Identifies a variable in text; replace this place holder with a specific value or string.
angle brackets	`<filename>`	Identifies a variable in code; replace this place holder with a specific value or string.
courier	`owsctl start wrb`	Text to be entered exactly as it appears. Also used for functions.

Convention	Example	Explanation
square brackets	`[-c string]`	Identifies an optional item.
	`[on|off]`	Identifies a choice of optional items, each separated by a vertical bar (|), any one option can be specified.
braces	`{yes|no}`	Identifies a choice of mandatory items, each separated by a vertical bar (|).
ellipses	n,...	Indicates that the preceding item can be repeated any number of times.

The term, Oracle Server, refers to the database server product from Oracle Corporation.

The term, **oracle**, refers to an executable or account by that name.

The term, *oracle*, refers to the owner of the Oracle software.

Oracle Services and Support

A wide range of information about Oracle products and global services is available from:

- **http://www.oracle.com**

The sections below provide URLs for selected services.

Oracle Support Services

Technical Support contact information worldwide is listed at:

- **http://www.oracle.com/support**

Templates are provided to help you prepare information about your problem before you call. You will also need your CSI number (if applicable) or complete contact details, including any special project information.

Product and Documentation

For U.S.A customers, Oracle Store is at:

- **http://store.oracle.com**

Links to Stores in other countries are provided from this site.

Product documentation can be found at:

- **http://docs.oracle.com**

Customer Service

Global Customer Service contacts are listed at:

- **http://www.oracle.com/support**

Education and Training

Training information and worldwide schedules are available from:

- **http://education.oracle.com**

Oracle Technology Network

Register with the Oracle Technology Network (OTN) at:

- **http://technet.oracle.com**

OTN delivers technical papers, code samples, product documentation, self-service developer support, and Oracle key developer products to enable rapid development and deployment of application built on Oracle technology.

1

Introduction to Oracle Internet Application Server

This chapter provides a general discussion of the Oracle Internet Application Server characteristics in comparison to those of Oracle Application Server (OAS). It includes a mapping of OAS components to their equivalent functionality in Oracle Internet Application Server.

Contents

What is Oracle Internet Application Server?

Oracle Internet Application Server is a middle-tier application server designed to enable scalability of web and database-centric applications beyond the limits of a single database instance. It offers:

- A deployment model with multiple deployment options.
- A variety of methods for generating web content, including PL/SQL and PSPs, Java servlets and JSPs, and Perl.
- Conformance to existing (and evolving) standards such as Java, J2EE, and CORBA.

OAS and Oracle Internet Application Server Components

The table below shows OAS components and their corresponding functionality in Oracle Internet Application Server.

Table 1–1 OAS and Oracle Internet Application Server functionality

OAS	Oracle Internet Application Server
JWeb application	Apache/JServ application
JServlet application	Apache/JServ application
LiveHTML application	mod_php application
Perl application	mod_perl application
JCORBA application	8*i* EJB application
ECO/Java application	8*i* EJB application
EJB application	8*i* EJB application
CWeb application	(no corresponding functionality at present)
PL/SQL application	mod_plsql

Enterprise Services Migration

This section discusses enterprise services, characteristics of a web site of concern to administrators and developers. It describes scalability, availability, fault tolerance, load balancing, and administration in OAS and how they will work after you migrate your site to Oracle Internet Application Server.

Overview

OAS consists of the HTTP layer, the server layer, and the application layer. The HTTP listener layer is made up of the HTTP server and the dispatcher. The Server layer provides a common set of components for managing these applications. These components include load balancing, logging, automatic failure recovery, security, directory, and transaction components. The application layer is made up of applications, cartridges, and cartridge servers. When a request arrives, the dispatcher routes the request to the application server layer and if a cartridge instance is available, the request will be serviced by that instance, otherwise a new instance will be created.

Similarly in Oracle Internet Application Server, the Apache Web Server (HTTP server), and mod_jserv run in the same process. Apache JServ is a pure Java servlet engine and runs in a separate process. The Apache Web Server uses mod_jserv to route requests to an Apache Jserv processes, much like the dispatcher in OAS.

Scalability

OAS can be deployed in single- or multi-node environments. Similarly, the Apache Web Server and Apache JServ can be configured for single or multi-node environments.

HTTP Server

In OAS, each listener can accommodate a maximum number of concurrent connections. This number varies based on operating system restrictions. To distribute the request load on a site, you can create multiple listeners, each listening on a different TCP port.

On Unix platforms, Apache Wed Server on start-up, creates a pool of child processes ready to handle incoming client requests. As the requests are processed and the load increases, Apache will make sure that there are a few extra processes running for subsequent requests. The initial and maximum size of the pool, and the min/max number of spare server processes, is configured with the StartServers, MaxClients, MinSpareServers and MaxSpareServers directives respectively.

On Windows NT, Apache runs as a multi-threaded process. The number of simultaneous connections is configured with the ThreadsPerChild directive, which is analogous to both the StartServers and MaxClients directives for UNIX.

Apache Web Server can be configured to run multiple instances on the same host, each of them using a different IP address/TCP port combination, or on different hosts.

Servlet Engine

In OAS, as the number of requests increases, the system creates new cartridge servers and new instances in them.

In Apache, mod_jserv receives requests from the Apache server and routes them to Apache JServ, the servlet engine.

Apache Jserv runs all servlets within servlet zones. Some of the advantages are: better security, run multiple JVMs and support for multiple virtual hosts.

Availability and Fault Tolerance

When a component such as a listener or a cartridge server fails, OAS detects the failure and restarts the failed component, restoring any preserved state information when possible.

In Apache, if there is more than one Apache server host, or more than one JServ host, and one of them stops, the system will still work as long as there is one Apache server and one JServ running. A last known status is maintained for every JServ, and any Apache server can route a request to any Apache JServ.

In Apache, the administrator is responsible for restarting any failed Apache Web Server or Apache Jserv instances.

Load Balancing

OAS allocates system resources and prioritizes requests based on two types of load balancing methods: priority-based and min / max.

In priority mode, the system manages and allocates resources automatically, based on the priority level you set for your applications and cartridges. The number of processes, threads, and instances is automatically determined based on the request load and priority level of the application and components.

In min / max mode, you set the number of instances, threads and client parameters for each cartridge at the cartridge level.

In Apache, you define the number of JServ hosts, host weight, and a logical set of these hosts in your configuration file. The system assigns incoming requests to JServs. If a JServ fails, requests are redirected to the other members of the logical set.

Administration

OAS provides GUI tools and built-in support for administering and monitoring your site, listeners, and applications.

In Apache, you perform site administration and maintenance by editing the Apache server and Apache JServ configuration files.

In Oracle Internet Application Server, as in OAS, GUI tools are provided for site administration and maintenance. The difference in the number and type of configuration files is significant:

OAS configuration file	Oracle Internet Application Server (Apache) configuration files
owl.cfg - list of registered OAS listeners and their configuration settings	**httpd.conf** - Primary (or sole) server-wide configuration file. (You have the choice of maintaining file location and translation information in **srm.conf**, and security information in **access.conf**—or maintaining all directives in one file.)
site.app - OAS site configuration file	
svl*istenerName*.cfg - listener configuration file	
wrb.app - process and cartridge configuration file	
resources.ora - configuration file for ORB	

Security

OAS supports a number of different security schemes for both user and host authentication, SSL, and the Oracle Wallet Manager.

In Oracle Internet Application Server, Apache JServ can run behind a firewall (the AJP protocol uses only one TCP port). It uses ACL (allowing AJP requests only from hosts with ACL) and supports SSL.

2

Migrating JWeb Applications to Apache JServ

This chapter discusses migration of JWEB applications from Oracle Application Server (OAS) to Apache JServ in the Oracle Internet Application Server. It includes a discusssion of functional differences between JWEB and JServ, and provides code examples for migrating.

Contents

What is Apache JServ?

Oracle Internet Application Server uses the Apache web server to service HTTP requests from clients. Apache JServ 1.1, a Servlet 2.0 compliant servlet engine, is bundled with Oracle Internet Application Server. If you have JWEB applications deployed on OAS 4.x and wish to migrate to Oracle Internet Application Server, you need to migrate your JWEB applications to Apache JServ 1.1.

Migrating OAS JServlets to Apache

Apache JServ 1.1 is compatible with Apache 1.3.x, JDK 1.1 or later, and JSDK 2.0. OAS 4.0.8.x JServlets are compliant with Servlet 2.1 specifications. If you are migrating OAS JServlets to Apache, we recommend that you plan the migration to Oracle8*i* JVM servlets. See **http://java.apache.org** for more information on Apache JServ.

JWEB and Apache JServ 1.1 Differences

This section describes the differences between JWEB and Apache JServ 1.1 applications.

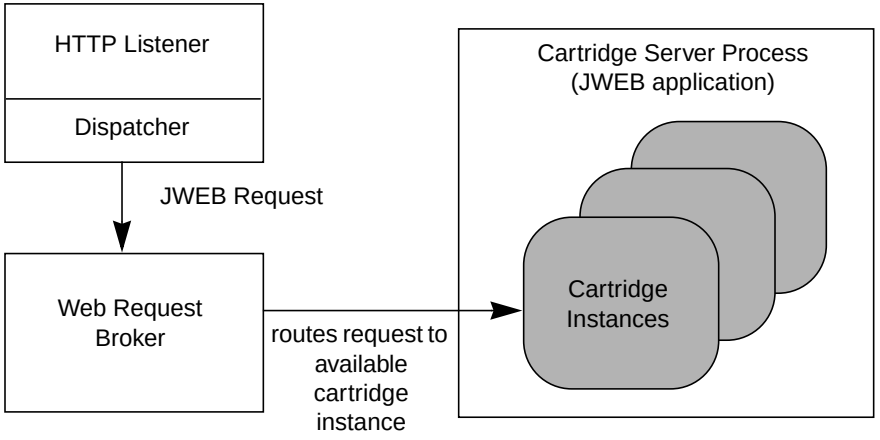
Architecture

JWEB applications execute within the OAS cartridge infrastructure, while Apache JServ 1.1 servlets run with the Apache web server and in JVM(s).

JWEB Architecture

In OAS, the HTTP listener receives a request for a JWEB cartridge. The listener passes the request to the dispatcher, which communicates with the Web Request Broker (WRB). The WRB uses URL mapping to identify the cartridge instance to which the request should be sent. If no cartridge instances exist for the requested cartridge, the cartridge server factory creates a cartridge server process to instantiate the cartridge. In JWEB, the cartridge server process loads a JVM, which runs a JWEB application (of the OAS application paradigm). Figure 2–1 depicts these components graphically.

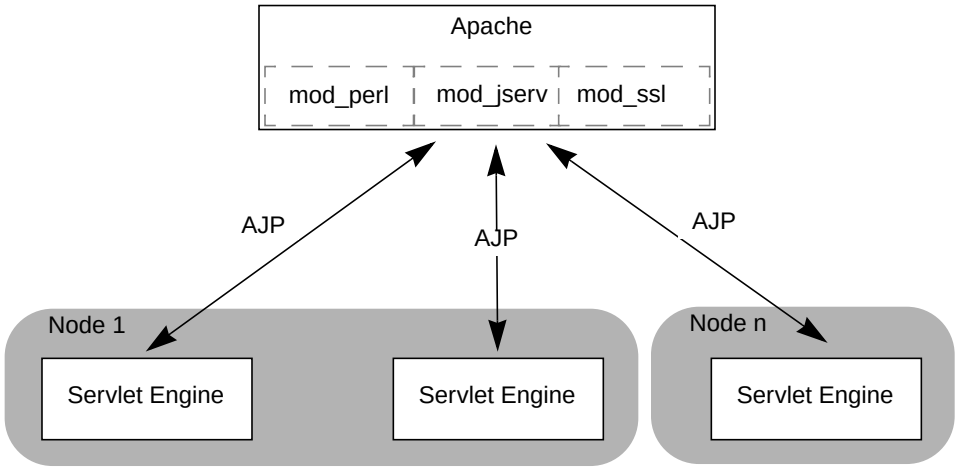
Figure 2–1 OAS Cartridge Infrastructure



Apache JServ Architecture

Apache JServ consists of two functional components: mod_jserv and a servlet engine, as shown in Figure 2–2.

Figure 2–2 Apache JServ Architecture (one-to-many example)



mod_jserv, which is implemented in C, is an Apache module that runs in the same process as the Apache web server. It functions like a dispatcher in that it receives a request from the Apache HTTP listener and routes it to a servlet engine. It does not execute any servlet business logic.

A servlet engine provides the runtime environment to execute servlets implementing the Servlet 2.0 API. It runs in a JVM process, in the same or different node as the Apache web server. Each JVM has one servlet engine, and the number of servlet engines is not proportional to the number of web servers (mod_jserv modules). One mod_jserv can work with more than one servlet engine and vice versa. Or, multiple mod_jserv modules can work with multiple servlet engines.

Apache JServ Protocol

Because Apache JServ servlet engines do not run in-process with mod_jserv (or possibly not even on the same physical machine as the module), a protocol is required for the two components to communicate. A proprietary protocol called Apache JServ Protocol (AJP) 1.1, is used. AJP 1.1 communicates using sockets, and implements an authentication algorithm using MD5 hashing without strong

cryptography. See **http://java.apache.org/jserv/protocol/AJPv11.html** for more information.

Single Node Configuration

When a servlet engine is located on the same machine as the web server, the mod_jserv module can be set up to start or stop the servlet engine and JVM when the web server starts or stops, respectively. The module performs all the necessary tasks to gracefully shut down the JVM. In this scenario, mod_jserv can also perform failover by checking JVM status regularly and starting another JVM if the first crashes.

Multi-Node Configuration

Automatic lifecycle control is not available when mod_jserv and a servlet engine exist on different machines. The engine and JVM must be started manually with a customizable script (each servlet engine requires its own script to start). This means that each engine can be started with a custom start-up environment.

The separation of the servlet engine from the web server makes Apache JServ scalable. The default limit on the number of servlet engines mod_jserv can address is 25. If you change this value, you need to recompile mod_jserv. You must consider hardware limitations when configuring the number of servlet engines to use in a site; each website has its own requirements for optimal configuration.

mod_jserv and servlet engine instances can have one-to-one, one-to-many, many-to-one, and many-to-many relationships. Multiple servlet engines can also reside on one node (in which case the JVMs must be assigned different port numbers so that mod_jserv can differentiate them).

Servlet Zones

Apache JServ implements a servlet virtualization paradigm called servlet zones. Servlet zones can be equated with virtual hosts of web servers. Each zone provides a logical demarcation from the physical relationships (locations) of servlet classes. Hence, each servlet zone can be assigned a common context, including a common URI, regardless of where its member servlets are located (for example, on different hosts). However, the current implementation of Apache JServ does not provide sandbox security for each zone.

Life cycle

JWEB classes and Apache JServ servlets have different life cycles.

JWEB Life Cycle

JWEB classes use the standard main() entry point to start their execution logic. Their life cycle resembles that of a standard Java class in loading, linking, initializing, and invoking main().

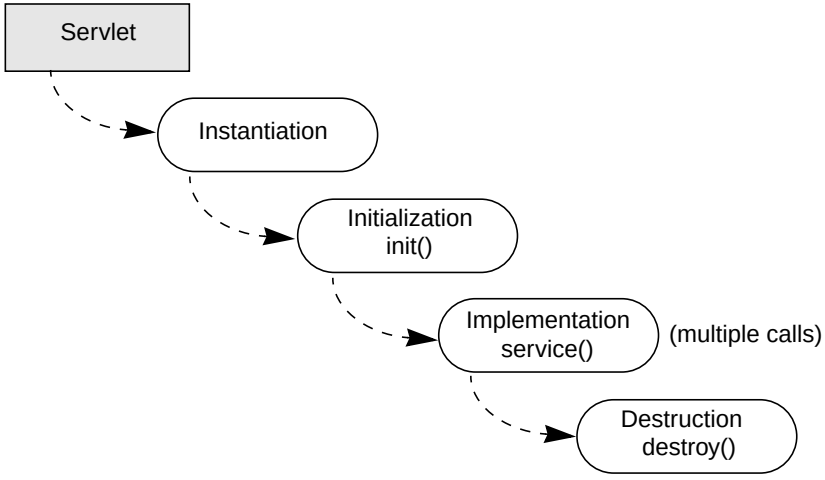
See http://java.sun.com/docs/books/vmspec/index.html for more details related to life cycle.

Apache JServ Life Cycle

In Apache JServ, servlet life cycle is compliant with Servlet 2.0 specifications. The life cycle is defined by the javax.servlet.Servlet interface, which is implemented directly or indirectly by all servlets. This interface has methods which are called at specific times by the servlet engine in a particular order during a servlet's lifecycle. The init() and destroy() methods are invoked once per servlet lifetime, while the service() method is called multiple times to execute the servlet's business logic.

Figure 2–3 depicts the servlet life cycle.

Figure 2–3 JServlet life cycle



Threading

The JWEB cartridge and Apache JServ servlet engine support single or multiple threads of execution, but the threading implementations are different.

JWEB Threading

Threading for the JWeb cartridge is defined in the OAS cartridge configuration by toggling the Stateless parameter (true or false). If true, a cartridge instance is shared by more than one client. If false, it is not shared, and only one client can access it at any one time. Also, if OAS is in min / max mode, the min / max cartridge servers and min / max threads values can be varied to change the way multi-threading is implemented for the cartridge.

Apache JServ Threading

The Apache JServ servlet engine is multi-threaded by default. The servlet container in the engine manages the threads that service client requests. Each instance of a servlet class can be given multiple threads of execution. In this case, a servlet instance is shared between more than one client. Alternatively, you can specify a class to execute only one thread at a time by having that class implement the `javax.servlet.SingleThread` interface. In this case, a pool of instances of this servlet class is maintained and each instance is assigned to one client only at any one time (instances are not shared).

Sessions

In the JWEB cartridge, you can enable client sessions using the OAS Node Manager. In Apache JServ, in accordance with Servlet 2.0 specifications, only programmable sessions are available. Consequently, if you are migrating a JWEB application that was session-enabled by a means other than code, you need to implement the session mechanism programmatically using the servlet session API. See "Session Control".

Dynamic Content Generation in HTML Pages

A JWEB Toolkit feature is available for generating dynamic content in HTML pages. The JWEB Toolkit embeds special placeholders in an HTML page. When this file is imported into a JWEB class as an oracle.html.HtmlFile object, the setItemAt() method places the data generated from the code at the placeholder locations.

Since this is a JWEB specific feature, it is not available in Apache JServ. If you would like to embed dynamic information in HTML pages (scripting), consider using Java Server Pages in Oracle Internet Application Server.

Code Modifications for JWEB Applications

To migrate JWEB applications to Apache JServ, you will have to modify code in these areas:

- Session Control
- Application Threads
- Logging

Session Control

You can session-enable a JWEB application with the cartridge's Client Session parameter in the Node Manager Web Parameters form. This allows the static parameters of an invoked class to contain per client data across calls. In the Servlet 1.0 API, session state is not kept in static variables of servlet classes. Instead, a session object is explicitly obtained to store session state using named attributes.

In Apache JServ, there is no configurable sessions support, so you have to enable sessions in code using the getSession()method in javax.servlet.http.HttpServletRequest, as shown below:

```
HttpSession session = request.getSession(true);
```

State information for a session can then be stored and retrieved by the putValue() and getValue() methods, respectively, of javax.servlet.http.HttpSession.

```
session.putValue("List", new Vector());
Vector list = (Vector) session.getValue("List");
```

Note: Do not use static data members to maintain session state in Apache JServ (although this is common practice in JWEB). Instead, use the servlet session API. The latter allows the servlet engine to use memory more efficiently.

JServ Session Time-out

You can specify the time-out value for a session in the session.timeout parameter in the jserv.properties file. You can also expire a session by invoking invalidate() in the servlet session API.

The JWEB session time-out callback is not available in Apache JServ.

Application Threads

In JWeb, an application can manage threads using the class:

oracle.owas.wrb.WRBRunnable

This class allows application threads to access request and response information. For Apache JServ, only standard Java thread management is needed to manage application threads (the java.lang.Runnable interface is used). For both JWeb and Apache JServ, using application threads is not recommended.

Logging

JWEB applications log messages using the OAS logger service provided by the WRB. This service allows applications to write messages to a central repository, such as a file system or database. The class:

oracle.owas.wrb.services.logger.OutputLogStream

interfaces with the logger service.

In Apache JServ, messages are written to two log files. Messages generated by mod_jserv are recorded in the file specified by the "ApJServLogFile" directive in the Apache web server's http.conf configuration file. The default value for this directive is <ServerRoot>/logs/mod_jserv.log. Messages generated by the servlet engine are recorded in the file specified by the log.file parameter in jserv.properties file (the default is <ServerRoot>/logs/jserv.log).

The messages generated by servlet applications, like exception stack traces, are recorded into jserv.log. In code, you can write to this log file using the javax.servlet.ServletContext.log or javax.servlet.GenericServlet.log methods.

The jserv.properties file allows you to select specific functional parts of the servlet engine to log. In jserv.log, these parts are referred to as channels.

Table 2–1 Channels in jserv.log

Channel	Type of Message
log.channel.authentication	Authentication messages from the AJP protocol.
log.channel.exceptionTracing	Exception stack traces caught by the servlet engine.
log.channel.init	Initialization messages from servlet engine.
log.channel.requestData	Data obtained from HTTP requests. For example, parameters from GET or POST HTTP methods.
log.channel.responseHeaders	Header information from HTTP responses.
log.channel.serviceRequest	Request processing messages.
log.channel.servletLog	Messages from the javax.servlet.ServletContext.log and javax.servlet.GenericServlet.log methods.
log.channel.servletManager	Messages from the servlet manager. These include messages for loading/unloading servlet zones and automatic class reloading.
log.channel.signal	System signal messages.
log.channel.terminate	Messages generated when servlet engine terminates.

JWEB Toolkit Packages (JWEB API)

The JWEB cartridge in OAS includes a JWEB toolkit of Oracle proprietary Java packages. If you used any of those packages in JWEB applications that will migrate to Oracle Internet Application Server, you must modify the code to use Servlet 2.0 equivalent classes and methods. If no equivalent functionality is available, you must rewrite the code to implement the functionality provided by the JWEB packages.

Because some of the JWEB toolkit packages were designed specifically to interact with OAS components such as the WRB, the functionality in these packages is not reproducible in the standard servlet API. Consequently, the migration process may also include some redesign of applications.

The following tables list JWEB methods and their functional equivalents for the following servlet API classes:

- Table 2–2 javax.servlet.http.HttpServletRequest
- Table 2–3 javax.servlet.ServletRequest
- Table 2–4 javax.servlet.ServletResponse
- Table 2–5 javax.servlet.ServletContext
- Table 2–6 javax.servlet.http.HttpUtils

Table 2–2 JWEB equivalents for javax.servlet.http.HttpServletRequest class methods

JWEB Method	Servlet Method
oracle.owas.wrb.services.http.HTTP.getHeader(String)	getHeader(name)
oracle.owas.wrb.services.http.getCGIEnvironment(“AUTH_TYPE”)	getAuthType()
oracle.owas.wrb.services.http.HTTP.getHeaders()[1]	getHeaderNames()[2]
oracle.owas.wrb.services.http.HTTP.getCGIEnvironment(“PATH_INFO”)	getPathInfo()
oracle.owas.wrb.services.http.HTTP.getCGIEnvironment(“PATH_TRANSLATED”)	getPathTranslated()
oracle.owas.wrb.services.http.HTTP.getCGIEnvironment(“QUERY_STRING”)	getQueryString()
oracle.owas.wrb.services.http.HTTP.getCGIEnvironment(“REQUEST_METHOD”)	getMethod()
oracle.owas.wrb.services.http.HTTP.getCGIEnvironment(“REMOTE_USER”)	getRemoteUser()
oracle.owas.wrb.services.http.HTTP.getCGIEnvironment(“SCRIPT_NAME”)	getServletPath()

[1] A hashtable of header names and values is returned.
[2] An enumeration of header names is returned.

Table 2–3 JWEB equivalents for javax.servlet.ServletRequest class methods

JWEB Method	Servlet Method
oracle.owas.wrb.services.http.HTTP.getCGIEnvironment(“CONTENT_TYPE”)	getContentType()
oracle.owas.wrb.services.http.HTTP.getCGIEnvironment(“CONTENT_LENGTH”)	getContentLength()
oracle.owas.wrb.services.http.HTTP.getCGIEnvironment(“SERVER_PROTOCOL”)	getProtocol()
oracle.owas.wrb.services.http.HTTP.getCGIEnvironment(“REMOTE_ADDR”)	getRemoteAddr()
oracle.owas.wrb.services.http.HTTP.getCGIEnvironment(“REMOTE_HOST”)	getRemoteHost()
oracle.owas.wrb.services.http.HTTP.getCGIEnvironment(“SERVER_NAME”)	getServerName()

Table 2–3 JWEB equivalents for javax.servlet.ServletRequest class methods

JWEB Method	Servlet Method
oracle.owas.wrb.services.http.HTTP.getCGIEnvironment("SERVER_PORT")	getServerPort()
oracle.owas.wrb.services.http.HTTP.getURLParameter(name)	getCharacterEncoding() getParameter(name)
oracle.owas.wrb.services.http.HTTP.getURLParameters	getParameterNames() getParameterValues(name)[1]

[1] where there are multiple values for "name"

Table 2–4 JWEB equivalents for javax.servlet.ServletResponse class methods

JWEB Method	Servlet Method
oracle.owas.wrb.WRBWriter	getWriter()

Table 2–5 JWEB equivalents for javax.servlet.ServletContext class methods

JWEB Method	Servlet Method
Use oracle.owas.wrb.services.http.HTTP.getCGIEnvironment to obtain the desired return value from this servlet method, if this value is available in the CGI environment.	getServerInfo()
Use oracle.OAS.Services.Logger	log(Exception, String) log(String)

Table 2–6 JWEB equivalents for javax.servlet.http.HttpUtils class methods

JWEB Method	Servlet Method
oracle.owas.wrb.services.http.HTTP.getURLParameters(Hashtable)	parsePostData(int, ServletInputStream)
oracle.owas.wrb.services.http.HTTP.getURLParameters(Hashtable)	parseQueryString(String)
oracle.html.HtmlStream.print	javax.servlet.ServletOutputStream.print
oracle.html.HtmlStream.println	avax.servlet.ServletOutputStream.println
oracle.owas.wrb.services.http.MultipartElement	javax.servlet.ServletInputStream.readLine

3

Migrating OAS Cartridges to Apache Modules

This chapter compares OAS cartridge functionality to corresponding functionality, and discusses considerations for migrating Oracle Application Server cartridges to the Apache Web Server infrastructure.

Contents

Cartridge Types and Corresponding Apache Modules

The table below shows the Apache infrastructure for each OAS cartridge type:

Table 3–1 Cartridge Types and Apache Modules

OAS Cartridge Type	Apache Module
Perl	mod_perl
LiveHTML	mod_php
PL/SQL	mod_plsql

The migration strategy for each application cartridge is detailed in the following sections.

PL/SQL Migration

OAS PL/SQL Cartridge applications can be migrated to Oracle Internet Application Server mod_plsql, which provides similar support for building and deploying PL/SQL based applications on the web.

mod_plsql is bundled into Oracle Internet Application Server as mod_plsql and is run as an Oracle HTTP Server module. It executes PL/SQL logic inside an Oracle database and delegates the servicing of HTTP requests to PL/SQL programs.

Users who are planning to migrate PL/SQL applications from OAS to Oracle Internet Application Server are encouraged to first read the Using mod_plsql documentation on the CD-ROM and familiarize themselves with the features in this product.

Support for the following OAS PL/SQL Cartridge features has changed in Oracle Internet Application Server mod_plsql. The rest of this section provides details on how to migrate OAS applications that use these features.

File Upload/Download

The following table lists the file upload/download features supported by OAS and Oracle Internet Application Server.

Table 3–2 File Upload/Download Features Comparison

File Upload/Download Feature	OAS Support	Oracle Internet Application Server Support
Upload/Download of file as raw byte streams without any character conversion	Yes	Yes
Upload of file into column type: LONG RAW	Yes	Yes
Upload of file into column type: BLOB	No	Yes
Upload of file into column type: CLOB, NCLOB	No	Yes
Specify per-application tables for upload of file	No - Uploads into WEBSYS scheme only	Yes
Compression/Decompression of file during file upload/download	Yes	No
Upload multiple files per form submission	Yes	Yes

Note that all OAS features except file compression / decompression are supported in Oracle Internet Application Server. Users with compressed uploaded files in OAS do not need to decompress their files manually. They will be automatically decompressed and uploaded in uncompressed format into the Oracle Internet Application Server document table by the **`oas2ias`** file migration tool that is documented in the next section.

For more information on the additional file upload features supported by Oracle Internet Application Server, refer to the Using mod_plsql documentation on your CD-ROM.

Uploaded File Document Format

OAS PL / SQL Cartridge and Oracle Internet Application Server mod_plsql both support uploaded files. However, they use different document table schemas. Users with uploaded files on OAS who wish to migrate to Oracle Internet Application Server will need to convert their files using the **oas2ias** migration tool.

The **oas2ias** tool performs two functions:

- Map data from the OAS tables to the Oracle Internet Application Server tables while maintaining the uploaded content and the content description.
- Deflate compressed content in OAS before migrating to Oracle Internet Application Server. This version of Oracle Internet Application Server does not support compression / decompression for uploaded files (see previous section).

The **oas2ias** tool is implemented in C, using the OCI library. The tool reads all the rows from the OWS_CONTENT table and populates the content and all it's attributes to a document table specified by the user.

The columns in the Oracle Internet Application Server document table will derive their values from OAS in the following way:

Table 3–3 Derived Column Values

Column in Oracle Internet Application Server doc table	OAS table.column value
NAME	ows_object.name
MIME_TYPE	ows_fixed_attrib.content_type
DOC_SIZE	ows_content.length
DAD_CHARSET	ows_fixed_attrib.character_set

Table 3–3 Derived Column Values

LAST_UPDATED	ows_object.last_modified
CONTENT_TYPE	“BLOB”
CONTENT	NULL
BLOB_CONTENT	OWS_CONTENT.content

The content from OAS will always be stored in the BLOB column of the Oracle Internet Application Server document table. The tool will also ensure that the data loaded into the Oracle Internet Application Server doc table is always uncompressed data. To do this, if the data is compressed (this is verified by checking the entries in the OWS_ATTRIBUTES table), the data is uncompressed using the zlib library, and then loaded to the document table in Oracle Internet Application Server.

Using the oas2ias Tool

The **oas2ias** tool need only be run once to convert all OAS files to Oracle Internet Application Server format. The following steps should be followed:

1. Make sure you have a current backup of all OAS uploaded files.
2. Create the document table for Oracle Internet Application Server. You can create this under any database user.

```
SQL> CREATE TABLE my_doc_table (
NAME VARCHAR2(128)UNIQUE NOT NULL,
MIME_TYPE      VARCHAR2(128),
DOC_SIZE       NUMBER,
DAD_CHARSET    VARCHAR2(128),
LAST_UPDATED   DATE,
CONTENT_TYPE   VARCHAR2(128),
CONTENT        LONG RAW,
BLOB_CONTENT   BLOB);
```

3. Verify the environment
   - OAS version 4.0.7.1 or later
   - iAS v1
   - Oracle database version 8.1.x

- ORACLE_HOME is set to Oracle Internet Application Server ORACLE_HOME
- ORACLE_HOME/bin is in the system path (on NT) or in the PATH environment variable (on UNIX)
- LD_LIBRARY_PATH contains $ORACLE_HOME/lib:/usr/java/lib

4. Setup TNS connect strings to the OAS database (where the "websys" schema exists) and the Oracle Internet Application Server database (where the Oracle Internet Application Server user schema with the `my_doc_table` table exists). Edit $ORACLE_HOME/network/admin/tnsnames.ora and setup the connect strings

```
<name> =
    (DESCRIPTION =
        (ADDRESS = (PROTOCOL = TCP)(Host = <hostname>)
        (Port = <portno>))
        (CONNECT_DATA = (SID = <sid>))
    )
```

5. Run the **oas2ias** tool which can be found in $ORACLE_HOME/bin in your Oracle Internet Application Server installation. The tool will then prompt for the following parameters:

Parameter	Description
websys_password	password for the "websys" user
websys_connstr	connect string for the OAS database
ias_user_name	database user name for the schema containing the Oracle Internet Application Server document table created in step 2
ias_password	password for <*ias_user_name*>
ias_connstr	connect string for the mod_plsql database
ias_doc_table	name of the Oracle Internet Application Server doc table created in step 2

The following is a sample run of **oas2ias**:

```
Welcome to the OAS to iAS migration Utility
Please enter the following parameters:
WEBSYS password: manager
OAS database connect string (<ENTER if local database>: orc8
iAS database user: oracle
```

```
iAS database user's password: welcome
iAS database connect string <ENTER if local database>: orc8
iAS doc table: my_doc_table

Transferred file : C:\TEMP\upload.htm
Length of file : 422
Transferred file : C:\Tnsnames.ora
Length of file : 2785
Transferred file : C:\rangan\mails1.htm
Length of file : 717835
Freeing handles ...
```

6. This completes the transfer of the files to an Oracle Internet Application Server document table and the files are now available for access using Oracle Internet Application Server mod_plsql.

Custom Authentication

Custom Authentication is used in OAS for applications that want to control the access themselves (that is within the application itself). The application authenticates the users in its own level and not within the database level.

mod_plsql supports custom authentication. See Using mod_plsql in the Oracle Internet Application Server Documentation Library for more information.

Positional Parameter Passing Scheme

The OAS PL/SQL cartridge supports a positional parameter passing scheme. This feature is not supported in Oracle Internet Application Server and cannot be used. Refer to the Using mod_plsql documentation on your CD-ROM for more information on the parameter passing schemes supported in Oracle Internet Application Server mod_plsql.

Executing SQL Files

In addition to running PL/SQL procedures stored in the database, the OAS PL/SQL cartridge can run PL/SQL source files from the file system. The source file contains an anonymous PL/SQL block that does not define a function or procedure. This feature enables users to execute PL/SQL statements without storing them in the database. This is useful when prototyping PL/SQL code since it saves having to reload procedures into the database each time they are edited.

This feature is not supported in Oracle Internet Application Server. Users will need to assign names to the anonymous blocks and compile them as stored procedures in the database.

Perl Migration

This section explains how Perl cartridge applications are implemented in the Oracle Application Server, and how they can be migrated to Oracle Internet Application Server.

Perl Applications under OAS

There are two types of Perl applications that can run under OAS: a Perl script run as a CGI script, and a Perl script run using the Perl cartridge.

Perl scripts that run under OAS as CGI scripts use a standard Perl interpreter that must be installed on the system as a Perl executable, separate from the OAS installation.

Perl scripts that run under OAS using the Perl cartridge use a Perl interpreter contained in the cartridge, and based on standard Perl version 5.004. The interpreter is built as a shared object in UNIX, libperlctx.so, and a shared library in NT, perlnt40.dll. The Perl cartridge links with the shared object or library at runtime.

Differences between Cartridge Scripts and CGI Scripts

Scripts written for the Perl cartridge differ from scripts written for a CGI environment, because of how the cartridge runs the interpreter. The Perl cartridge:

- Maintains a persistent interpreter, and pre-compiles and caches Perl scripts (thus achieving better performance).
- Redirects stdin and stdout to the WRB client I/O (i.e., the browser).
- Redirects stderr to the WRB logger.
- Returns additional CGI environment variables to the Perl interpreter whenever it calls for system environment variables.
- Supports the system call instead of the fork call. The system call modifies the implementation of the Perl interpreter to redirect child process output to the WRB client I/O.
- Supports error logging.

- Supports performance instrumentation.

Perl scripts developed under OAS to run as CGI scripts can run in Oracle Internet Application Server as CGI scripts without modification. However, Perl scripts developed to run under the Perl cartridge in OAS may need to be modified to run under Oracle Internet Application Server.

Migrating Perl Cartridge Scripts from OAS to Oracle Internet Application Server

This section includes a discussion of OAS and Oracle Internet Application Server Perl implementations, and code modifications for migrating Perl scripts to Oracle Internet Application Server.

The Oracle Internet Application Server Perl Environment

The Oracle Internet Application Server Perl environment is based on the Apache Perl distribution (mod_perl). Like the OAS implementation, mod_perl provides a persistent Perl interpreter embedded in the server and a code caching feature that loads and compiles modules and scripts only once, serving them from the cache. Like the OAS Perl cartridge, mod_perl redirects stdout to the listener.

See www.apache.org for more information on mod_perl.

Installation Requirements

The Perl DBI and DBD modules are not part of the standard Oracle Internet Application Server distribution, and must be installed separately. Refer to the release notes for details on version requirements, download links, and installation instructions.

Perl Version

Perl version 5.004_04 or higher must be installed before mod_perl can be installed.

Perl Modules

The table below lists Perl modules shipped with OAS. These modules are not a part of the standard Oracle Internet Application Server distribution, so in order to migrate applications that use these modules from OAS to Oracle Internet Application Server, you must acquire these modules from the listed sources and install them.

Table 3–4 Perl modules in OAS

Module	Version
DBI	0.79
DBD::Oracle	0.44
LWP or libwww-perl	5.08
CGI	2.36
MD5	1.7
IO	1.15
NET	1.0502
Data-Dumper	2.07

Variations from OAS Perl Cartridge

The following points should be noted between the OAS Perl cartridge and Oracle Internet Application Server mod_perl.

Namespace Collision

In OAS and Oracle Internet Application Server, compiled Perl scripts are cached. If not properly handled, the caching of multiple Perl scripts can lead to namespace collisions. To avoid this, both OAS and Oracle Internet Application Server translate the Perl script file name into a unique packaging name, and then compile the code into the package using eval. The script is then available to the Perl application in compiled form, as a subroutine in the unique package name.

OAS and Oracle Internet Application Server form the package name differently. OAS cannot cache subroutines with the same name. Oracle Internet Application Server creates the package name by prepending Apache::ROOT:: and the path of the URL (substituting "::" for " / ").

Using cgi-lib.pl

OAS Perl scripts that use cgi-lib.pl must be modified to use a version of the library customized for the Perl cartridge. (For more information on cgi-lib.pl, see http://cgi-lib.stanford.edu/cgi-lib.) This is not necessary for Oracle Internet Application Server.

Refer to the release notes to find out how to modify these Perl scripts to run in Oracle Internet Application Server.

Pre-loading modules

OAS Perl scripts may contain instructions that need not be executed repetitively for each request of the script. Performance improves if these instructions are run only once, and only the necessary portion is run for each request of the Perl script.

In OAS, the **$ORAWEB_HOME/../cartx/common/perl/lib/perlinit.pl** file is used to pre-load modules and perform initial tasks. This file is executed only once when the cartridge instance starts up. By default, there are no executable statements in this file. This file is specified by the Initialization Script parameter in the Perl Cartridge Configuration form.

There is no corresponding functionality in Apache mod_perl.

LiveHTML Migration

In Oracle Application Server, you can generate dynamic content using the LiveHTML cartridge by embedding Server-Side Includes (SSI) and scripts in HTML pages. If you are migrating LiveHTML applications to Oracle Internet Application Server, you need to migrate LiveHTML SSI to Apache SSI and LiveHTML embedded scripts to mod_php scripts.

SSI

SSIs provided by Apache and LiveHTML do not have equivalent elements. The following table lists the SSIs available in Apache and LiveHTML.

Table 3–5 List of SSIs in Apache and LiveHTML

Apache SSIs	LiveHTML SSIs
config	config
echo	echo
exec	exec

Table 3–5 List of SSIs in Apache and LiveHTML

Apache SSIs	LiveHTML SSIs
fsize	fsize
flastmod	flastmod
include	include
printenv	-
set	-
-	request

The syntax for specifying an SSI in Apache or LiveHTML is the same. For example:

```
<!--#config sizefmt="bytes" -->
```

(Note that a space is required before the closing terminator "`-->`".)

SSI in Apache is implemented by the mod_include module. This module is compiled into the Apache Server by default.

In addition to the elements shown in the table above, Apache SSI also includes variable substitution and flow control elements. Refer to the documentation at http://www.apache.org.

Scripts

Scripts in LiveHTML are written in Perl. For PHP scripts, the syntax used takes from C and Perl. As a detailed discussion of the differences between these two types of scripts is outside the scope of this document, refer to the appropriate documentation for LiveHTML/Perl and PHP.

PHP information can be found at http://php.apache.org and Perl at http://www.perl.com.

4

Migrating EJB, ECO/Java and JCORBA Applications

This chapter provides information on migrating EJB, ECO for Java and JCO applications from the Oracle Application Server to Oracle8*i* JVM EJB objects. A working knowledge of deploying EJBs on Oracle8*i* JVM is assumed, as that subject is not discussed here.

Contents

Migrating EJBs

To migrate EJBs from OAS 4.0.8 (or later) to Oracle8*i* JVM, you will need to modify code in the following areas:

- Deployment descriptors
- Client code
- Logging (server code) (if applicable)

These changes are described in the following sections.

Deployment Descriptors

Oracle8*i* JVM allows you to put deployment information in a text file that you can run through the ejbdescriptor command line tool to create the serialized

deployment descriptors. The format of this text file resembles Java. The example below shows the basic structure, in which `ejb.test.server` is the package that contains the implementation of the bean class `ExampleBean`.

```
SessionBean ejb.test.server.ExampleBeanImpl
{
   <attribute>=<value>
   ...
}
```

The required attributes are:

- BeanHomeName
- HomeInterfaceClassName
- RemoteInterfaceClassName

Common additional attributes include:

- StateManagementType (to define whether the bean is stateful or stateless)
- TransactionAttribute (to set the transaction attribute value)
- RunAsMode (to specify the privileges allowed to the bean)
- RunAsIdentity (to specify the privileges allowed to the bean)
- AllowedIdentities (to state who has access to the bean)

You can also use this format to set Java environment variables for the bean. For additional information, see the *Oracle 8i EJB and CORBA Developer's Guide*.

Note: You can also use serialized deployment descriptors instead of the text file.

Client Code

Changes to the client code are made in the initial context call using JNDI. The hashtable passed to the initial context call must contain all of the properties listed in the table below.

Table 4–1 Hashtable values

Property	Value
javax.naming.Context.URL_PKG_ PREFIXES	oracle.aurora.jndi
javax.naming.Context.SECURITY_ AUTHORIZATION	oracle.aurora.sess_iiop.ServiceCtx.NON_SSL_ LOGIN **or** oracle.aurora.sess_iiop.ServiceCtx.SSL_ CREDENTIAL **or** oracle.aurora.sess_iiop.ServiceCtx.SSL_LOGIN
javax.naming.Context.SECURITY_ PRINCIPAL	The database or Oracle 8*i* Cache username, for example, Scott
javax.naming.Context.SECURITY_ CREDENTIALS	The user password, for example, Tiger

You must also change the URL that accesses your EJB home to the Oracle 8*i* format:

sess_iiop://<host>:<port>:<SID>/<path>/<bean>

For example:

sess_iiop://myhost:2481:ORCL/test/myBean

Logging (Server Code)

If application logging was done in OAS, remove all references to **oracle.oas.ejb.Logger** from your EJB code. In Oracle8*i* JVM, you can use the `println` function for simple logging, or you can log to the database.

Migrating ECO/Java

When migrating ECO for Java to Oracle 8*i*, you can choose between migrating to EJB, or to CORBA. As the ECO model is very similar to EJB, the easiest migration is to EJB. You will need to change server code as described in the sections below in addition to changes for deployment descriptors and client code described in the sections above for EJB migration.

To modify your ECO for Java components to be compatible with Oracle8*i* JVM EJBs, you must modify the implementation file, the remote interface file, and the home interface file.

Remote Interface

Change the remote interface to extend `javax.ejb.EJBObject` instead of `oracle.oas.eco.ECOObject`. Each method must throw `java.rmi.RemoteException`.

Home Interface

Change the home interface to extend `javax.ejb.EJBHome` instead of `oracle.oas.eco.ECOHome`.

The create method must throw `javax.ejb.CreateException` and `java.rmi.RemoteException` instead of `oracle.oas.eco.CreateException`.

Implementation Class

Make the following changes to the implementation class:

1. Remove all occurrences of, and references to, `oracle.oas.eco.Logger`.
2. Change all occurrences of `oracle.oas.eco.* to javax.ejb.*`.
3. Change ECOCreate method to ejbCreate method.
4. Change ECORemove method to ejbRemove method.
5. Change ECOActivate method to ejbActivate method.
6. Change ECOPassivate method to ejbPassivate method.

Migrating JCORBA to EJB

Oracle Application Server versions 4.0.6 and 4.0.7 provided a component model called Java CORBA Objects (JCO), a precursor to the ECO for Java model. This section discusses migrating from JCO in OAS to EJB in Oracle8*i* JVM.

To migrate to EJB, you must modify the server and client code as discussed in the following sections. To modify the server code, you must modify the remote interface, create a home interface, modify the JCORBA object implementation, and make parameters serializable. You must also modify the deployment descriptors as discussed in "Migrating EJBs".

Remote Interface

Make the following changes to the remote interface:

1. Convert all occurrences of `org.omg.CORBA.Object` or `oracle.oas.jco.JCORemote` to `javax.ejb.EJBObject`.
2. Throw `java.rmi.RemoteException` for all methods in the interface.

Home Interface

You will need to create a home interface as defined in the EJB specification. An example is shown below.

```
import javax.ejb.*;
import java.rmi.RemoteException;
public interface ServerStackHome extends EJBHome
{
  public ServerStackRemote create() throws CreateException, RemoteException;
}
```

Object Implementation

Complete the following to migrate the implementation class:

1. Change import `oracle.oas.jco.*` to import `javax.ejb.*`.
2. Check that the class implements `javax.ejb.SessionBean`.

> **Note:** The JCORBA Lifecycle is not supported within EJB; if the JCORBA object implements `oracle.oas.jco.Lifecycle`, you must remove it.

3. Remove any logger references.
4. Move any initialization operations to the `ejbCreate()` method.
5. Save the session context passed into the `setSessionContext()` method in an instance variable.
6. Ensure that all public methods in the class throw the `java.rmi.RemoteException` exception.
7. Change any ObjectManager type to SessionContext type. The table below maps the methods in the ObjectManager class to methods in the SessionContext class.

Table 4–2 ObjectManager and SessionContext methods

SessionContext Method	ObjectManager Method
getEnvironment()	getEnvironment()
Parameter passed to setSessionContext()	getObjectManager()
getEJBObject()	getSelf()
getEJBObject().remove()	revokeSelf()
getUserTransaction()	getCurrentTransaction()

Make Parameters Serializable

If any user defined parameters are being passed in the remote interface, ensure that the classes implement `java.io.Serializable`.

Index

L

M

N

O

P

R

S

T

U

W