Oracle9*i* Application Server

Migrating from Oracle Application Server

Release 1.0.2

November 2000

Part No. A83709-04

Oracle9*i* Application Server Migrating from Oracle Application Server, Release 1.0.2

Part No. A83709-04

Primary Author: Sanjay Singh

Contributors: Kai Li, Beth Roeser, Susan Gordon, Matthieu Devin, Pushkar Kapasi

Contents

Send Us Your Comments

Oracle9*i* Application Server Release 1.0.2, Migrating from Oracle Application Server

Part No. A83709-04

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:

- Electronic mail - iasdocs_us@oracle.com
- Fax - (650) 654-6206 Attn: Oracle9*i* Application Server Documentation Manager
- Postal service:

  Oracle Corporation
  Oracle9*i* Application Server Documentation Manager
  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 guide describes the process of migrating your system from Oracle Internet Application Server Release 1.0.0 to Oracle9*i* Application Server Release 1.0.2.

This preface contains these topics:

Audience

This guide is for system administrators and application developers who will be migrating their system from Oracle Application Server to Oracle9*i* Application Server.

To use this document, you need to be familiar with the configuration, operation, and development of Oracle Application Server and other system administration tasks.

Organization

This document contains:

Chapter 1, "Introduction to Oracle Internet Application Server"

This chapter provides an introduction to Oracle9*i* Application Server and migration options for Oracle Application Server users.

Chapter 2, "Migrating JWeb Applications to Apache JServ"

This chapter discusses migration options for Oracle Application Server JWeb Cartridge users.

Chapter 3, "Migrating Oracle Application Server Cartridges"

This chapter discusses the migration options for the other Oracle Application Server cartridge types including the PL/SQL cartridge.

Chapter 4, "Migrating EJB, ECO/Java and JCORBA Applications"

This chapter discusses the migration options for the Oracle Application Server IIOP components.

Related Documentation

For more information, see these Oracle resources:

- *Oracle9i* Application Server Documentation Library CD-ROM
- Oracle9*i* Application Server Platform Specific Documentation on Oracle9*i* Application Server Disk 1

In North America, printed documentation is available for sale in the Oracle Store at

`http://oraclestore.oracle.com/`

Customers in Europe, the Middle East, and Africa (EMEA) can purchase documentation from

`http://www.oraclebookshop.com/`

Other customers can contact their Oracle representative to purchase printed documentation.

To download free release notes, installation documentation, white papers, or other collateral, please visit the Oracle Technology Network (OTN). You must register online before using OTN; registration is free and can be done at

`http://technet.oracle.com/membership/index.htm`

If you already have a username and password for OTN, then you can go directly to the documentation section of the OTN Web site at

`http://technet.oracle.com/docs/index.htm`

For additional information, see:

- *Third-Party Book* by (insert first and last names of authors). (insert name of publisher), (insert publication date).

(In this section, list all third-party documentation, including Web sites, that you refer to in the document. References to Web sites should use Type 13 and Type 14 markers so that the URL will become a link in HTML when the file is filtered.)

Conventions

This section describes the conventions used in the text and code examples of the this documentation set. It describes:

- Conventions in Text
- Conventions in Code Examples

Conventions in Text

We use various conventions in text to help you more quickly identify special terms. The following table describes those conventions and provides examples of their use.

Convention	Meaning	Example
Bold	Bold typeface indicates terms that are defined in the text or terms that appear in a glossary, or both.	The C datatypes such as **ub4**, **sword**, or **OCINumber** are valid. When you specify this clause, you create an **index-organized table**.
Italics	Italic typeface indicates book titles, emphasis, syntax clauses, or placeholders.	*Oracle8i Concepts* You can specify the *parallel_clause*. Run `U`*`old_release`*`.SQL` where *old_release* refers to the release you installed prior to upgrading.
UPPERCASE monospace (fixed-width font)	Uppercase monospace typeface indicates elements supplied by the system. Such elements include parameters, privileges, datatypes, RMAN keywords, SQL keywords, SQL*Plus or utility commands, packages and methods, as well as system-supplied column names, database objects and structures, user names, and roles.	You can specify this clause only for a `NUMBER` column. You can back up the database using the `BACKUP` command. Query the `TABLE_NAME` column in the `USER_TABLES` data dictionary view. Specify the `ROLLBACK_SEGMENTS` parameter. Use the `DBMS_STATS.GENERATE_STATS` procedure.
lowercase monospace (fixed-width font)	Lowercase monospace typeface indicates executables and sample user-supplied elements. Such elements include computer and database names, net service names, and connect identifiers, as well as user-supplied database objects and structures, column names, packages and classes, user names and roles, program units, and parameter values.	Enter `sqlplus` to open SQL*Plus. The `department_id`, `department_name`, and `location_id` columns are in the `hr.departments` table. Set the `QUERY_REWRITE_ENABLED` initialization parameter to `true.` Connect as `oe` user.

Conventions in Code Examples

Code examples illustrate SQL, PL/SQL, SQL*Plus, or other command-line statements. They are displayed in a monospace (fixed-width) font and separated from normal text as shown in this example:

```
SELECT username FROM dba_users WHERE username = 'MIGRATE';
```

The following table describes typographic conventions used in code examples and provides examples of their use.

Convention	Meaning	Example
[ ]	Brackets enclose one or more optional items. Do not enter the brackets.	`DECIMAL (digits [ , precision ])`
{ }	Braces enclose two or more items, one of which is required. Do not enter the braces.	`{ENABLE | DISABLE}`

|	A vertical bar represents a choice of two or more options within brackets or braces. Enter one of the options. Do not enter the vertical bar.	`{ENABLE | DISABLE}`
		`[COMPRESS | NOCOMPRESS]`

...	Horizontal ellipsis points indicate either:	
	■ That we have omitted parts of the code that are not directly related to the example	`CREATE TABLE ... AS subquery;`
	■ That you can repeat a portion of the code	`SELECT col1, col2, ... , col`*`n`*` FROM employees;`
. . .	Vertical ellipsis points indicate that we have omitted several lines of code not directly related to the example.	

Other notation	You must enter symbols other than brackets, braces, vertical bars, and ellipsis points as it is shown.	`acctbal NUMBER(11,2);`
		`acct    CONSTANT NUMBER(4) := 3;`

`Italics`	Italicized text indicates variables for which you must supply particular values.	`CONNECT SYSTEM/`*`system_password`*

`UPPERCASE`	Uppercase typeface indicates elements supplied by the system. We show these terms in uppercase in order to distinguish them from terms you define. Unless terms appear in brackets, enter them in the order and with the spelling shown. However, because these terms are not case sensitive, you can enter them in lowercase.	`SELECT last_name, employee_id FROM employees;`
		`SELECT * FROM USER_TABLES;`
		`DROP TABLE hr.employees;`

`lowercase`	Lowercase typeface indicates programmatic elements that you supply. For example, lowercase indicates names of tables, columns, or files.	`SELECT last_name, employee_id FROM employees;`
		`sqlplus hr/hr`

1 Introduction to Oracle Internet Application Server

This chapter provides a general discussion of the Oracle9*i* Application Server characteristics in comparison to those of Oracle Application Server. It includes a mapping of Oracle Application Server components to their equivalent functionality in Oracle9*i* Application Server. The topics include:

- What is Oracle9i Application Server?
- Oracle Application Server Component Migration Options
- Enterprise Services Migration

What is Oracle9*i* Application Server?

Oracle9*i* 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.

Oracle Application Server Component Migration Options

The table below shows Oracle Application Server components and their corresponding functionality in Oracle9*i* Application Server.

Table 1–1 Comparison of Application Models

Oracle Application Server	Oracle9*i* Application Server
JWeb application	Apache JServ application
JServlet application	Apache JServ application
LiveHTML application	Apache SSI and OracleJSP applications
Perl application	mod_perl application
JCORBA application	Oracle8*i* JVM EJB application
ECO/Java application	Oracle8*i* JVM EJB application
EJB application	Oracle8*i* JVM EJB application
CWeb application	Custom Apache Modules, CGI, Java JNI and PL/SQL Callouts
PL/SQL application	mod_plsql application

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 Oracle Application Server and how they will work after you migrate your site to Oracle9*i* Application Server.

Overview

Oracle Application Server 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 Oracle9*i* Application Server, the Oracle 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 process, much like the dispatcher in Oracle Application Server.

Scalability

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

HTTP Server

In Oracle Application Server, 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, Oracle HTTP Server creates a pool of child processes ready to handle incoming client requests, on start-up. As the requests are processed and the load increases, the server spawn new processes 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, Oracle HTTP Server 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.

You can configure Oracle HTTP Server 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 Oracle Application Server, as the number of requests increases, the system creates new cartridge servers and new instances in them.

In Oracle HTTP Server, mod_jserv receives requests from the 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, the ability to 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, Oracle Application Server detects the failure and restarts the failed component, restoring any preserved state information when possible.

In Oracle HTTP Server, if there is more than one server host, or more than one JServ host, and one of them stops, the system will still work as long as there is one server and one JServ running. A last known status is maintained for every JServ, and any Oracle HTTP Server instance 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

Oracle Application Server 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 Oracle HTTP Server, 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 JServ instances. If a JServ instance fails, requests are redirected to the other members of the logical set.

Administration

Oracle Application Server provides GUI tools and built-in support for administering and monitoring your site, listeners, and applications. The configuration data from the OAS Manager tool is stored in various configuration files.

In Oracle HTTP Server, you perform site administration and maintenance by editing the Apache server and Apache JServ configuration files. The difference from Oracle Application Server in the number and type of configuration files is significant.

Table 1–2 Configuration Files

Oracle Application Server Listener	Oracle HTTP Server (Apache)
`owl.cfg` - list of registered 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` - site configuration file	(no equivalent)
`sv`*`listenerName`*`.cfg` - listener configuration file	(no equivalent)
`wrb.app` - process and cartridge configuration file	(no equivalent)
`resources.ora` - configuration file for ORB	(no equivalent)

Security

Oracle Application Server 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 to Apache JServ in the Oracle9*i* Application Server. It includes a discusssion of functional differences between JWeb and Apache JServ, and provides code examples for migrating. The topics include:

What is Apache JServ?

Oracle9*i* Application Server uses the Oracle HTTP Server to service HTTP requests from clients. Apache JServ 1.1, a Servlet 2.0 compliant servlet engine, is bundled with Oracle9*i* Application Server. If you have JWeb applications deployed on Oracle Application Server 4.x and wish to migrate to Oracle9*i* Application Server, you need to migrate your JWeb applications to the Servlet 2.0 specification.

Migrating Oracle Application Server JServlets to Apache JServ Servlets

Apache JServ 1.1 is compatible with Apache 1.3.x, JDK 1.1 or later, and JSDK 2.0. Oracle Application Server 4.0.8 JServlets are compliant with the Servlet 2.1 specification. If you are migrating JServlets to Apache, we recommend that you plan the migration to Oracle8*i* JVM servlets because of the differences between 2.0 and 2.1 compliant servlets.

See Also:

- `http://java.apache.org` for more information on Apache JServ
- `http://java.sun.com` for more information on the Servlet specifications

JWeb and Apache JServ 1.1 Differences

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

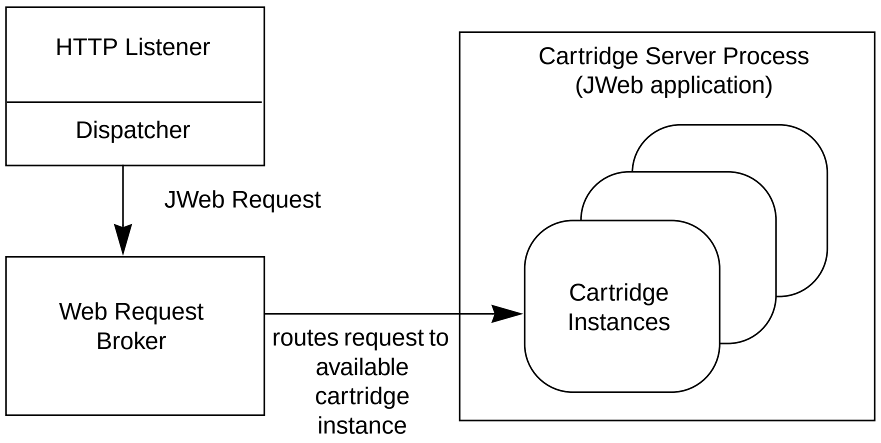
Architecture

JWeb applications execute within the Oracle Application Server cartridge infrastructure, while Apache JServ 1.1 servlets run with the Oracle HTTP Server and in JVM(s).

JWeb Architecture

In Oracle Application Server, 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 Oracle Application Server application paradigm). Figure 2–1 depicts these components graphically.

Figure 2–1 Oracle Application Server Cartridge Infrastructure

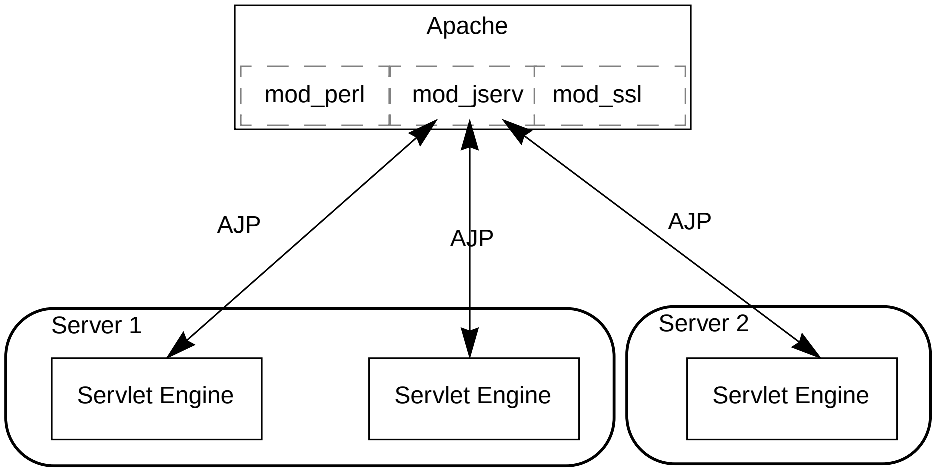

HTTP Listener
Dispatcher
JWeb Request
Web Request Broker
routes request to available cartridge instance
Cartridge Server Process (JWeb application)
Cartridge Instances

Apache JServ Architecture

Apache JServ consists of two functional components: mod_jserv and a servlet engine. mod_jserv is an Apache Server module and directs incoming requests for Java Servlets to a servlet engine. The Apache JServ Protocol (AJP) facilitates communication between the two components.

Figure 2–2 illustrates a one-to-many configuration. In a one-to-many configuration, there is one Apache listener and multiple servers. Each server can run one or more servlet engines. In this figure, a single Apache instance is communicating to two servers. Server 1 is running two servlet engines and server 2 is running one servlet engine. Three AJP connections are open between the servlet engines and a single mod_jserv in the Apache instance.

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 Also:

`http://java.apache.org/jserv/protocol/AJPv11.html`

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 environment. There is a limit of 25 servlet engines that can be addressed by a single mod_jserv.

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 Also:

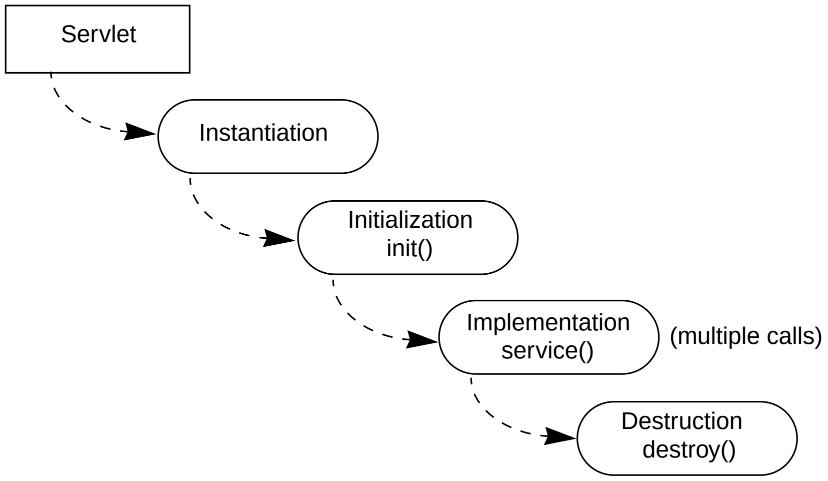
`http://java.sun.com/docs/books/vmspec/index.html`

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 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 Oracle Application Server 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 Oracle Application Server 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 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" on page 2-8.

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 JavaServer Pages with OracleJSP in Oracle9*i* 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 `oracle.owas.wrb.WRBRunnable` class. 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 because multi-threaded applications limit the effectiveness of the load balancer.

Logging

JWeb applications log messages using the Oracle Application Server 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 `oracle.owas.wrb.services.logger.OutputLogStream` class 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 Oracle HTTP Server `http.conf` configuration file. The default value for this directive is `<ORACLE_HOME>/Apache/Jserv/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 value for this directive is `<ORACLE_HOME>/Apache/Jserv/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.

Table 2–1 Channels in jserv.log

Channel	Type of Message
log.channel.terminate	Messages generated when servlet engine terminates.

JWeb Toolkit Packages (JWeb API)

The JWeb cartridge in Oracle Application Server includes a JWeb toolkit of Oracle proprietary Java packages. If you used any of those packages in JWeb applications that will migrate to Oracle9*i* 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 Oracle Application Server 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, "JWeb Equivalents for javax.servlet.http.HttpServletRequest Class Methods"
- Table 2–3, "JWeb Equivalents for javax.servlet.ServletRequest Class Methods"
- Table 2–4, "JWeb Equivalents for javax.servlet.ServletResponse Class Methods"
- Table 2–5, "JWeb Equivalents for javax.servlet.ServletContext Class Methods"
- Table 2–6, "JWeb Equivalents for javax.servlet.http.HttpUtils Class Methods"

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()

Table 2–2 JWeb Equivalents for javax.servlet.http.HttpServletRequest Class Methods

JWeb Method	Servlet Method
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()
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
oracle.owas.wrb.services.http.HTTP.getCGIEnvironment	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 Oracle Application Server Cartridges

This chapter compares Oracle Application Server cartridge functionality to corresponding functionality, and discusses considerations for migrating cartridges to the Oracle9*i* Application Server infrastructure. The topics include:

Cartridge Types and Corresponding Apache Modules

The table below shows the Oracle HTTP Server equivalent for each Oracle Application Server cartridge type:

Table 3–1 Cartridge Types and Apache Modules

Oracle Application Server Cartridge Type	Oracle HTTP Server Equivalent
Perl	mod_perl
LiveHTML	Apache SSI and OracleJSP
PL/SQL	PL/SQL Gateway
CWeb	Custom Apache Modules, CGI, Java JNI and PL/SQL Callouts

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

PL/SQL Migration

Oracle Application Server PL/SQL Cartridge applications can be migrated to Oracle9*i* Application Server PL/SQL Gateway, which provides similar support for building and deploying PL/SQL based applications on the web.

mod_plsql is bundled into the PL/SQL Gateway and runs as an Oracle HTTP Server module. It delegates the servicing of HTTP requests to PL/SQL programs which execute their logic inside of Oracle databases.

Users who are planning to migrate PL/SQL applications from Oracle Application Server to Oracle9*i* Application Server are encouraged to read *Using the PL/SQL Gateway* in the Oracle9*i* Application Server Documentation Library and familiarize themselves with the features in this module.

Support for the following Oracle Application Server PL/SQL Cartridge features has changed in Oracle9*i* Application Server PL/SQL Gateway. The rest of this section provides details on how to migrate Oracle Application Server applications that use these features.

File Upload/Download

The following table lists the file upload / download features supported by Oracle Application Server and Oracle9*i* Application Server.

Table 3–2 File Upload/Download Features Comparison

File Upload/Download Feature	Oracle Application Server Support	Oracle9*i* 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 tables for upload of file for each database access descriptor (DAD)	No - Uploads into WEBSYS schema only	Yes
Compression / Decompression of file during file upload / download	Yes	No
Upload multiple files per form submission	Yes	Yes

Note that all Oracle Application Server features except file compression / decompression are supported in Oracle9*i* Application Server. Users with compressed uploaded files in Oracle Application Server do not need to decompress their files manually. They will be automatically decompressed and uploaded in uncompressed format into the Oracle9*i* Application Server document table by the `oas2ias` file migration tool that is documented in "Using the oas2ias Tool" on page 3-5.

See Also: *Using the PL/SQL Gateway* in the Oracle9*i* Application Server Documentation Library for more information on additional file upload features.

Uploaded File Document Format

Oracle Application Server PL / SQL Cartridge and Oracle9*i* Application Server PL / SQL Gateway both support uploaded files. However, they use different document table schemas. Users with uploaded files on Oracle Application Server who wish to migrate to Oracle9*i* Application Server will need to convert their files using the oas2ias migration tool.

The oas2ias tool performs two functions:

- Mapping data from the Oracle Application Server tables to the Oracle9*i* Application Server tables while maintaining the uploaded content and the content description.
- Deflating compressed content in Oracle Application Server before migrating to Oracle9*i* Application Server. This version of Oracle9*i* Application Server does not support compression / decompression for uploaded files (see the previous section for further details).

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.

Table 3–3 shows how the columns in the Oracle9*i* Application Server document table derive their values from Oracle Application Server.:

Table 3–3 Derived Column Values

Column in Oracle Internet Application Server Document Table	Oracle Application Server 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
LAST_UPDATED	ows_object.last_modified
CONTENT_TYPE	“BLOB”
CONTENT	NULL
BLOB_CONTENT	OWS_CONTENT.content

The content from Oracle Application Server will always be stored in the BLOB_CONTENT column of the Oracle9*i* Application Server document table. The tool will also ensure that the data loaded into the Oracle9*i* 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 Oracle9*i* Application Server.

Using the oas2ias Tool

The `oas2ias` tool need only be run once to convert all Oracle Application Server files to Oracle9*i* Application Server format. The following steps should be followed:

1. Make sure you have a current backup of all Oracle Application Server uploaded files.
2. Create the document table for Oracle9*i* 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
   - Oracle Application Server Release 4.0.7.1 or later
   - Oracle9*i* Application Server Release 1.0.0 or later
   - Oracle database version 8.1.x
   - ORACLE_HOME is set to Oracle9*i* Application Server ORACLE_HOME
   - (Windows only) The system path contains `%ORACLE_HOME%\bin`
   - (UNIX only) The PATH environment variable contains `$ORACLE_HOME/bin`
   - (UNIX only) The LD_LIBRARY_PATH environment variable contains both `$ORACLE_HOME/lib and /usr/java/lib`
4. Create TNS aliases to the Oracle Application Server database (where the websys schema exists) and the Oracle9i Application Server database (where the Oracle9i Application Server user schema with the my_doc_table table exists). Store the aliases in `$ORACLE_HOME/network/admin/tnsnames.ora` (UNIX)

or `%ORACLE_HOME%\network\admin\tnsnames.ora` (Windows NT). The format for a TNS alias in this file is:

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

See your database documentation for more information on TNS aliases.

5. Run the `oas2ias` tool which can be found in the bin directory under ORACLE_HOME in your Oracle9*i* Application Server installation. The tool will prompt for the following parameters:

Parameter	**Description**
websys_password	password for the `websys` user
websys_connstr	connect string for the Oracle Application Server 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 PL/SQL Gateway database
ias_doc_table	name of the Oracle9*i* 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 Oracle9*i* Application Server document table and the files are now available for access using Oracle9*i* Application Server PL/SQL Gateway.

Custom Authentication

Custom Authentication is used in Oracle Application Server 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.

The PL/SQL Gateway also supports custom authentication.

See Also: *Using the PL/SQL Gateway* in the Oracle9*i* Application Server Documentation Library for more information on authentication

Flexible Parameter Passing

The flexible parameter passing scheme allows you to overload PL/SQL procedures. This allows you to reuse the same procedure name but change the procedure's behaviour depending on how many parameters a form passes to the procedure.

Both Oracle Application Server and Oracle9*i* Application Server support flexible parameter passing. To use flexible parameter passing in the PL/SQL Gateway, prefix the procedure name with an exclamation point (!) in the invoking URL.

For example, if the following URL invokes your Oracle Application Server procedure:

`http://`*<host>*`/`*<virtual_path>*`/procedure?x=1&y=2`

Then the URL that invokes your PL/SQL Gateway procedure will be:

`http://`*<host>*`/`*<virtual_path>*`/!procedure?x=1&y=2`

See Also: *Using the PL/SQL Gateway* in the Oracle9*i* Application Server Documentation Library for more information flexible parameter passing

Positional Parameter Passing

The Oracle Application Server PL/SQL cartridge supports a positional parameter passing scheme. This feature is not supported in Oracle9*i* Application Server and cannot be used.

See Also: *Using the PL/SQL Gateway* in the Oracle9*i* Application Server Documentation Library for more information on supported parameter passing schemes

Executing SQL Files

In addition to running PL/SQL procedures stored in the database, the Oracle Application Server 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 Oracle9*i* 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 Oracle9*i* Application Server.

Perl Applications under Oracle Application Server

There are two types of Perl applications that can run under Oracle Application Server:

- Perl scripts running as a CGI scripts
- Perl scripts using the Perl cartridge

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

Perl scripts that run under Oracle Application Server 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 either:

- (UNIX only) `libperlctx.so`, a shared object
- (Windows only) `perlnt40.dll`, a shared library

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 Oracle Application Server to run as CGI scripts can run in Oracle9*i* Application Server as CGI scripts without modification. However, Perl scripts developed to run under the Perl cartridge in Oracle Application Server may need to be modified to run under Oracle9*i* Application Server.

Migrating Perl Cartridge Scripts

This section includes a discussion of Oracle Application Server and Oracle9*i* Application Server Perl implementations, and code modifications for migrating Perl scripts to Oracle9*i* Application Server.

The Oracle9*i* Application Server Perl Environment

The Oracle9*i* Application Server Perl environment is based on the Apache Perl distribution (mod_perl). Like the Oracle Application Server 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 Oracle Application Server Perl cartridge, mod_perl redirects stdout to the listener.

See Also: the mod_perl documentation in the Oracle9*i* Application Server Documentation Library

Installation Requirements

The Perl DBI and DBD modules are not part of the standard Oracle9*i* Application Server distribution, and must be installed separately. Refer to the Release Notes for details on version requirements, download sites, and installation instructions.

Perl Modules

Table 3–4 lists Perl modules shipped with Oracle Application Server. These modules are not a part of the standard Oracle9*i* Application Server distribution. In order to migrate applications that use these modules from Oracle Application Server to Oracle9*i* Application Server, you must acquire these modules and install them. The files are available from:

```
http://www.cpan.org
```

Table 3–4 Perl Modules in Oracle Application Server

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 Oracle Application Server Perl Cartridge

The following points should be noted between the Oracle Application Server Perl cartridge and Oracle9*i* Application Server mod_perl.

Namespace Collision

In Oracle Application Server and Oracle9*i* 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 Oracle Application Server and Oracle9*i* 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.

Oracle Application Server and Oracle9*i* Application Server form the package name differently. Oracle Application Server cannot cache subroutines with the same name. Oracle9*i* Application Server creates the package name by prepending `Apache::ROOT::` and the path of the URL (substituting "`::`" for "`/`").

Using cgi-lib.pl

Oracle Application Server Perl scripts that use `cgi-lib.pl` must be modified to use a version of the library customized for the Perl cartridge. This is not necessary for Oracle9*i* Application Server.

See Also: ■

- `http://cgi-lib.stanford.edu/cgi-lib` for more information on `cgi-lib.pl`
- Oracle9*i* Application Server Release Notes for information about modifying `cgi-lib.pl`

Pre-loading modules

Oracle Application Server 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 Oracle Application Server, `perlinit.pl` pre-loads modules and performs 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 Oracle9*i* Application Server, you need to migrate LiveHTML SSI to Apache SSI. Currently the only equivalent to LiveHTML embedded scripts in Oracle9*i* Application Server is JavaServer Pages.

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
fsize	fsize

Table 3–5 List of SSIs in Apache and LiveHTML (Cont.)

Apache SSIs	LiveHTML SSIs
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: The space before the closing terminator (**--**>) is required.

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.

See Also: the Apache Server documentation in the Oracle9*i* Application Server Documentation Library

Scripts

In Oracle Application Server, you can use the LiveHTML cartridge to embed Perl scripts in HTML files. Because LiveHTML is a proprietary Oracle Application Server component, it does not have equivalent functionality in Oracle9*i* Application Server. However, JavaServer Pages allow you to embed Java code in HTML files. The JavaServer Pages 1.0 (JSP) model is implemented in Oracle9*i* Application Server as OracleJSP.

To migrate your LiveHTML application to Oracle9*i* Application Server, you must do the following:

- Migrate from the LiveHTML application model to the JSP application model.
- Migrate LiveHTML tags to JSP tags.
- Rewrite the Perl code as Java code.

If your LiveHTML application uses Web Application Objects in Oracle Application Server, you must implement this functionality as embedded Java code, or as JavaBean classes, and declare them with the `<jsp:useBean>` tag in JSPs.

> **See Also:** *Oracle JavaServer Pages Developer's Guide and Reference* in the Oracle9*i* Application Server Documentation Library

CWeb Migration

In Oracle Application Server, you can use the CWeb (or C) Cartridge to:

- create custom cartridges
- develop applications that other cartridges invoke

There is no simple migration path from Oracle Application Server CWeb Cartridges to Oracle9*i* Application Server. If you used CWeb to create custom cartridges you should consider creating a custom Apache module.

If you use CWeb to invoke C programs, you have the following options:

- CGI scripts: standalone C programs generating web content with `println` statements.
- Java JNI: Java Servlets or JavaServer Pages that call C routines from Apache JServ or Oracle8*i* JVM
- PL/SQL callouts: PL/SQL applications that call C routines from Oracle Database Cache or Oracle8*i*.

The Web Request Broker (WRB) and C APIs are not available in Oracle9*i* Application Server.

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. Oracle8i JVM is the Oracle9*i* Application Server component that provides a runtime environment for EnterpriseJava Bean applications. The topics include:

- Migrating EJBs
- Migrating ECO / Java
- Migrating JCORBA to EJB

Migrating EJBs

To migrate EJBs from Oracle Application Server 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)

The following sections describe these changes.

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.

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

See Also: *Oracle8i Enterprise JavaBeans and CORBA Developer's Guide* in the Oracle9*i* Application Server documentation library

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`	One of: ▪ `oracle.aurora.sess_iiop.ServiceCtx.NON_SSL_LOGIN` ▪ `oracle.aurora.sess_iiop.ServiceCtx.SSL_CREDENTIAL` ▪ `oracle.aurora.sess_iiop.ServiceCtx.SSL_LOGIN`
`javax.naming.Context.SECURITY_PRINCIPAL`	The database or Oracle Database 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 of:

```
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 Oracle Application Server, 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 (ECO / Java) to Oracle8*i* JVM, 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 (ECO / Java) model. This section discusses migrating from JCO in Oracle Application Server 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 "Deployment Descriptors" on page 4-2.

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