Oracle9*i* Application Server

Migrating from Oracle Application Server

Release 2 (9.0.2)

January 2002

Part No. A95108-01

ORACLE®

Oracle9*i* Application Server Migratiing from Oracle Application Server, Release 2 (9.0.2)

Part No. A95108-01

Contributors: Min Yao, Sanjay Singh, Kai Li, Beth Roser, Sheryl Maring, Baogang Song, Song Lin, and Stephen Mayer

Contents

Send Us Your Comments

Oracle9*i* Application Server Migrating from Oracle Application Server, Release 2 (9.0.2)

Part No. A95108-01

Oracle Corporation welcomes your comments and suggestions on the quality and usefulness of this document. 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?

If you find any errors or have any other suggestions for improvement, please indicate the document title and part number, and 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: 650-506-7407 Attn: Oracle9*i* Application Server Documentation Manager
- Postal service:
  Oracle Corporation
  Oracle9*i* Application Server Documentation
  500 Oracle Parkway, M/S 2op3
  Redwood Shores, CA 94065
  USA

If you would like a reply, please give your name, address, telephone number, and (optionally) electronic mail address.

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 Application Server to Oracle9*i* Application Server.

This preface contains these topics:

- Intended Audience
- Documentation Accessibility
- Organization
- Related Documentation
- Conventions

Intended Audience

Migrating from Oracle Application Server is intended for system administrators and application developers who will migrate their systems 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.

Documentation Accessibility

Our goal is to make Oracle products, services, and supporting documentation accessible, with good usability, to the disabled community. To that end, our documentation includes features that make information available to users of assistive technology. This documentation is available in HTML format, and contains markup to facilitate access by the disabled community. Standards will continue to evolve over time, and Oracle Corporation is actively engaged with other market-leading technology vendors to address technical obstacles so that our documentation can be accessible to all of our customers. For additional information, visit the Oracle Accessibility Program Web site at

`http://www.oracle.com/accessibility/`

Accessibility of Code Examples in Documentation JAWS, a Windows screen reader, may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, JAWS may not always read a line of text that consists solely of a bracket or brace.

Accessibility of Links to External Web Sites in Documentation This documentation may contain links to Web sites of other companies or organizations that Oracle Corporation does not own or control. Oracle Corporation neither evaluates nor makes any representations regarding the accessibility of these Web sites.

Organization

This document contains:

Chapter 1, "Introduction to Oracle9i Application Server"

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

Chapter 2, "Migrating JWeb & JServlet Applications to OC4J"

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:

- Oracle9*i* 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://otn.oracle.com/admin/account/membership.html`

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://otn.oracle.com/docs/index.html`

Conventions

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

- Conventions in Text
- Conventions in Code Examples
- Conventions for Microsoft Windows Operating Systems

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.	When you specify this clause, you create an **index-organized table**.
Italics	Italic typeface indicates book titles or emphasis.	*Oracle9i Database Concepts* Ensure that the recovery catalog and target database do *not* reside on the same disk.
`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, usernames, and roles.	You can specify this clause only for a NUMBER column. You can back up the database by using the BACKUP command. Query the TABLE_NAME column in the USER_TABLES data dictionary view. Use the DBMS_STATS.GENERATE_STATS procedure.

Convention	Meaning	Example
`lowercase monospace (fixed-width) font`	Lowercase monospace typeface indicates executables, filenames, directory names, 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, usernames and roles, program units, and parameter values. **Note:** Some programmatic elements use a mixture of UPPERCASE and lowercase. Enter these elements as shown.	Enter `sqlplus` to open SQL*Plus. The password is specified in the `orapwd` file. Back up the datafiles and control files in the `/disk1/oracle/dbs` directory. 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. The `JRepUtil` class implements these methods.
`lowercase italic monospace (fixed-width) font`	Lowercase italic monospace font represents placeholders or variables.	You can specify the *`parallel_clause`*. Run `U`*`old_release`*`.SQL` where *`old_release`* refers to the release you installed prior to upgrading.

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]`

Convention	Meaning	Example
`. . .`	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`*`, ... ,` *`coln`* `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 shown.	`acctbal NUMBER(11,2);`
		`acct    CONSTANT NUMBER(4) := 3;`

`Italics`	Italicized text indicates placeholders or variables for which you must supply particular values.	`CONNECT SYSTEM/`*`system_password`*
		`DB_NAME =` *`database_name`*

`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.
	Note: Some programmatic elements use a mixture of UPPERCASE and lowercase. Enter these elements as shown.
		`SELECT last_name, employee_id FROM employees;`
		`sqlplus hr/hr`
		`CREATE USER mjones IDENTIFIED BY ty3MU9;`

Conventions for Microsoft Windows Operating Systems

The following table describes conventions for Microsoft Windows operating systems and provides examples of their use.

Convention	Meaning	Example
Choose Start >	How to start a program.	To start the Oracle Database Configuration Assistant, choose Start > Programs > Oracle - *HOME_NAME* > Configuration and Migration Tools > Database Configuration Assistant.
File and directory names	File and directory names are not case sensitive. The following special characters are not allowed: left angle bracket (<), right angle bracket (>), colon (:), double quotation marks ("), slash (/), pipe (|), and dash (-). The special character backslash (\\) is treated as an element separator, even when it appears in quotes. If the file name begins with \\\\, then Windows assumes it uses the Universal Naming Convention.	`c:\winnt"\"system32` is the same as `C:\WINNT\SYSTEM32`
`C:\>`	Represents the Windows command prompt of the current hard disk drive. The escape character in a command prompt is the caret (^). Your prompt reflects the subdirectory in which you are working. Referred to as the *command prompt* in this manual.	`C:\oracle\oradata>`

	The backslash (\\) special character is sometimes required as an escape character for the double quotation mark (") special character at the Windows command prompt. Parentheses and the single quotation mark (’) do not require an escape character. Refer to your Windows operating system documentation for more information on escape and special characters.	`C:\>exp scott/tiger TABLES=emp QUERY=\"WHERE job=’SALESMAN’ and sal<1600\"`
		 `C:\>imp SYSTEM/`*`password`*` FROMUSER=scott TABLES=(emp, dept)`

`HOME_NAME`	Represents the Oracle home name. The home name can be up to 16 alphanumeric characters. The only special character allowed in the home name is the underscore.	`C:\> net start Oracle`*`HOME_NAME`*`TNSListener`

Convention	Meaning	Example
ORACLE_HOME and *ORACLE_ BASE*	In releases prior to Oracle8*i* release 8.1.3, when you installed Oracle components, all subdirectories were located under a top level *ORACLE_HOME* directory that by default used one of the following names: ■ `C:\orant` for Windows NT ■ `C:\orawin95` for Windows 95 ■ `C:\orawin98` for Windows 98 This release complies with Optimal Flexible Architecture (OFA) guidelines. All subdirectories are not under a top level *ORACLE_HOME* directory. There is a top level directory called *ORACLE_BASE* that by default is `C:\oracle`. If you install Oracle9*i* release 1 (9.0.1) on a computer with no other Oracle software installed, then the default setting for the first Oracle home directory is `C:\oracle\ora90`. The Oracle home directory is located directly under *ORACLE_BASE*. All directory path examples in this guide follow OFA conventions. Refer to *Oracle9i Database Getting Starting for Windows* for additional information about OFA compliances and for information about installing Oracle products in non-OFA compliant directories.	Go to the *ORACLE_BASE*`\`*ORACLE_ HOME*`\rdbms\admin` directory.

1

Introduction to Oracle9*i* Application Server

This chapter provides a general discussion of the Oracle9*i* Application Server (Oracle9*i*AS) 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*AS. 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*AS provides full support for the Java 2 Enterprise Platform (J2EE), XML, and emerging Web services standards. With Oracle9*i*AS you can simplify information access for your customers and trading partners by delivering enterprise portals, which can be customized and accessed from a network browser or wireless devices. It allows you to redefine your business processes, and integrate your applications and data sources with those from your customers or partners. You can deliver tailored customer experiences via real-time personalization, and assess and correlate Web site traffic patterns using Oracle9*i*AS integrated business intelligence services.

You can also implement a centralized management, security, and directory framework to manage and monitor all of your distributed systems and diverse user communities. Oracle9*i*AS allows you to save on Web site infrastructure by deploying your fast, scalable Internet applications through built-in Web caching, load balancing and clustering capabilities.

> **See Also:** *Oracle9i Application Server Concepts* in Oracle9*i*AS documentation library for additional information on Oracle9*i*AS

Before proceeding to migration, you must have successfully deployed the Oracle9*i*AS product and have worked with the examples provided in respective sections of *Oracle9i Application Server Concepts* and other related documentation.

Oracle Application Server Component Migration Options

Table 1–1 presents Oracle Application Server components and their corresponding functionality in Oracle9*i*AS. During the migration process, you must migrate these Oracle Application Server components to their closest corresponding components in Oracle9*i*AS. Please refer to the reference chapters presented in Table 1–1 for detailed discussions on specific components.

Table 1–1 Comparison of Application Components

Oracle Application Server Component	Closest Oracle9*i* Application Server Equivalent Component	Reference
JWeb application	Oracle9*i*AS Container for J2EE (OC4J) application	Chapter 2
JServlet application	OC4J application	Chapter 2
LiveHTML application	Apache SSI and JavaServer Page (JSP) applications	Chapter 3
Perl application	`mod_perl` application	Chapter 3
CWeb application	Custom Apache Modules, Common Gateway Interface (CGI), FastCGI, Java Naming and Directory Interface (JDNI), and PL/SQL Callouts	Chapter 3
PL/SQL application	`mod_plsql` application	Chapter 3
ECO/Java application	OC4J application	Chapter 4
EJB application	OC4J application	Chapter 4
JCORBA application	OC4J application	Chapter 4

Enterprise Services Migration

This section discusses enterprise services and characteristics of a Web site of concern to administrators and developers. It describes scalability, availability, fault tolerance, load balancing, administration, security, and the third-party Web server support in Oracle Application Server. It also describes whether migrating your Web site from Oracle Application Server to Oracle9*i*AS affects these characteristics.

Overview

Oracle Application Server consists of three layers, the HTTP listener layer, the server layer, and the applications layer. The HTTP listener layer consists of listeners, the adapter interface, and the dispatchers. The server layer provides a common set of components for managing applications. These components include load balancing, logging, automatic failure recovery, security, directory, and transaction management components. The application layer consists 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.

> **See Also:** *Oracle Application Server Overview and Glossary* for details on Oracle Application Server.

In Oracle9*i*AS, Oracle HTTP Server handles load-balancing, routing servlet requests to OC4J through `mod_oc4j`, single sign-on authentication and security context propagation through `mod_osso` and SSL. OC4J consists of pure J2EE containers for running JSPs, Servlets, and Enterprise JavaBeans (EJBs), and provides J2EE container services. Both the Oracle HTTP Server and OC4J perform the same functions as three layers in Oracle Application Server.

Scalability

You can deploy Oracle Application Server in single or multiple-host environments. You can configure the Oracle HTTP Server and OC4J for single or clustered-host environments.

HTTP Server

In Oracle Application Server, each listener accommodates 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.

For Oracle9*i*AS on UNIX platforms, Oracle HTTP Server creates a pool of child processes ready to handle incoming client requests during the start-up. As the requests load increases, the server spawns new processes for subsequent requests. The initial and maximum size of the pool, and the minimum or maximum number of spare server processes is configured with the `StartServers`, `MaxClients`, `MinSpareServers` and `MaxSpareServers` directives, respectively.

For Oracle9*i*AS on Windows platforms, 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 Application Server through the Node Manager. For Oracle9*i*AS, you can configure Oracle HTTP Server using Oracle Enterprise Management (OEM) graphical user interfaces (GUIs), or by manually editing the `http.conf` file.

> **See Also:** *Oracle HTTP Server Administration Guide* and *Oracle9i Application Server Administrator's Guide* in the Oracle9*i*AS Documentation Library

OC4J Container

In Oracle Application Server, as the number of requests increases, the system creates new cartridge servers and new instances.

In Oracle9*i*AS Oracle HTTP Server, `mod_oc4j` receives requests from the server and routes them to the OC4J servlet container.

> **See Also:** *Oracle9iAS Containers for J2EE User's Guide* in the Oracle9*i*AS Documentation Library

Refer to Chapter 2, "Migrating JWeb & JServlet Applications to OC4J" and Chapter 4, "Migrating EJB, ECO/Java and JCORBA Applications" for Migration of Oracle Application Server components to Oracle9*i*AS OC4J.

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, and restoring any preserved state information, when possible.

In Oracle HTTP Server, if there is more than one HTTP server host, or more than one OC4J host, when one of the hosts stops, the system will still function as long as one HTTP server and one OC4J are running, provided that J2EE components have been deployed against the cluster of OC4J instances. Any Oracle HTTP Server instance can route a request to any OC4J instance. Maintaining routing information in cookies eliminates single point of failure.

Load Balancing

Oracle Application Server allocates system resources and prioritizes requests based on two types of load balancing methods, priority-based method and minimum or maximum-based method.

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 minimum or maximum mode, you set the number of instances, threads and client parameters for each cartridge at the cartridge level.

In Oracle HTTP Server, you can define the number of hosts and a logical set of these hosts in your configuration file. The system assigns incoming requests to OC4J instances.

Configuration of an instance determines whether the instance is part of a cluster. If an OC4J instance may be part of one cluster, all of its configured components are implicitly part of that cluster. A cluster can contain one or more instances. Each installation can have only one instance. There can be many installations on one host.

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 Oracle Application Server Manager tool is stored in various configuration files.

In Oracle HTTP Server, you can perform site administration and maintenance using OEM through GUIs, or through a set of configuration files. Table 1–2 presents configuration files for the Oracle Application Server HTTP listener and Oracle HTTP Server. The configuration files between two servers are significantly different.

See Also: *Oracle9i Application Server Administrator's Guide* and *Oracle HTTP Server Administration Guide* in the Oracle9*i*AS Documentation Library

Table 1–2 Configuration Files

Oracle Application Server HTTP Listener	Oracle9*i*AS Oracle HTTP Server
`owl.cfg` – list of registered listeners and their configuration settings	`httpd.conf` – primary (or sole) server-wide configuration file (You can choose to maintain file location and translation information in `srm.conf`, and security information in `access.conf`, or to maintain 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 the ORB	(no equivalent)

Security

You must convert the certificate from Oracle Application Server to Oracle9*i*AS.

Migrating Certificates

Oracle9*i*AS contains two migration tools, pconvert and `ssl2ossl` (Unix) or `osslconvert` (Windows). You can take the following two steps to migrate from the Oracle Application Server certificate to an Oracle9*i*AS certificate or wallet.

1. Convert the Oracle Application Server private key to an Oracle9*i*AS private key using the conversion tool, `pconvert`. The full path to the tool is:
   - (UNIX) *`ORACLE_HOME/`*`Apache/Apache/bin/pconvert`
   - (Windows) *`ORACLE_HOME`*`\Apache\Apache\bin\pconvert.exe`

   The syntax for running `pconvert` is:

   ```
   pconvert -s oas_private_key_file -d ias_private_key_file
   ```

   For example:

   ```
   prompt> pconvert -s privkey.der -d iaskey.pem
   ```

2. Generate an Oracle9*i*AS wallet file using the Oracle Application Server certificate file and the `ias_private_key` file that you obtained from step 1 with the conversion tool, `ssl2ossl` or `osslconvert`. The full paths to the tools are:

   - (UNIX) *`ORACLE_HOME/`*`Apache/Apache/bin/ssl2ossl`
   - (Windows) *`ORACLE_HOME`*`\Apache\Apache\bin\osslconvert.exe`

   The syntax for running `ssl2ossl` on Unix is:

```
ssl2ossl -cert oas_certificate_file
         -key ias_private_key_file
         -wltpass password_for_wallet
         -certpass password_for_oas_certificate_file
         -chain oas_certificate_chain_file
         -capath oas_certificate_authority_path
         -cafile oas_certificate_authority_file
         -wallet wallet_full_path
         -ssowallet yes/no
         -validate yes/no
```

   The syntax for running `osslconvert` on Windows is:

```
osslconvert.exe -cert oas_certificate_file
                -key ias_private_key_file
                -wltpass password_for_wallet
                -certpass password_for_oas_certificate_file
                -chain oas_certificate_chain_file
                -capath oas_certificate_authority_path
                -cafile oas_certificate_authority_file
                -wallet wallet_full_path
                -ssowallet yes/no
                -validate yes/no
```

   Table 1–3 summarizes the parameters and their associated requirements for the `ssl2ossl` or `osslconvert` conversion tool.

Table 1–3 Summary of `ssl2ossl` or `osslconvert` Tool Parameters

Parameter	Description	Requirement
`cert`	your Oracle Application Server certificate file	required
`key`	the *`ias_private_key_file`* you just obtained from step 1	required
`certpass`	the password for the certificate	optional

Table 1–3 Summary of* `ssl2ossl` *or* `osslconvert` *Tool Parameters

Parameter	Description	Requirement
`wltpass`	the password for the wallet	optional
`chain`	the Oracle Application Server certificate chain file	optional, but of chain, capath, or cafile, at least one of these parameters are required.
`capath`	the Oracle Application Server certificate authority path	optional, but of chain, capath, or cafile, at least one of these parameters are required.
`cafile`	the Oracle Application Server certificate authority file	optional, but of chain, capath, or cafile, at least one of these parameters are required.
`wallet`	the full path of your wallet file	optional, but the default path is: `ORACLE_HOME/Apache/Apache/conf/ssl.wlt/0`
`ssowallet`	with a value of either `yes` or `no`	optional, the default value is `no`
`validate`	with a value of either `yes` or `no` - If `yes`, then the tool will not generate a wallet. - If `no`, the tool will generate a wallet.	optional, the default value is `no`.

See Also: *Oracle9i Application Server Security Guide* in the Oracle9*i*AS Documentation Library for details on `ssowallet` and other security information

Third-Party Web Server Support

Oracle9*i*AS uses Oracle HTTP Server and Oracle Application Server uses HTTP Server as their Web listeners. However, many companies only use Microsoft Internet Information Services (IIS) or iPlanet as their corporate standard Web server.

Both Oracle Application Server and Oracle9*i*AS support the third-party Web servers, such as IIS and iPlanet.

2

Migrating JWeb & JServlet Applications to OC4J

This chapter discusses migration of JWeb and JServlet applications from Oracle Application Server to OC4J in the Oracle9*i*AS. Topics include:

- JWeb and OC4J Differences
- Migration Strategies
- Code Modifications for JWeb Applications

JWeb and OC4J Differences

This section provides background information on JWeb and OC4J. It also describes the differences between JWeb and OC4J applications.

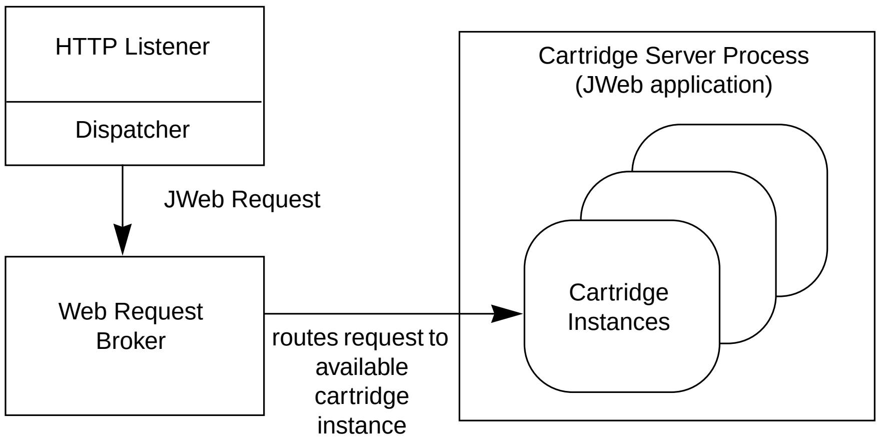
Architecture

JWeb applications execute within the Oracle Application Server cartridge infrastructure, while OC4J runs on a standard Virtual Machine.

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 Security or Web Request Broker (WRB). The WRB uses a 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

OC4J Architecture

OC4J consists of a Web container including servlet and JSP engines, EJB container, J2EE services APIs (JNDI, JTA, JMS, and JAAS), and enterprise information systems APIs (JDBC, SQLJ, J2EE connector architecture).

`mod_oc4j` is a dynamically loaded module of Oracle HTTP Server with the purpose of routing requests through Oracle HTTP Server to OC4J processes. `mod_oc4j` takes into account OC4J sessions information to route requests back to the original OC4J process and to re-route failed session requests to other members of the same OC4J Island when the original OC4J process is unreachable.

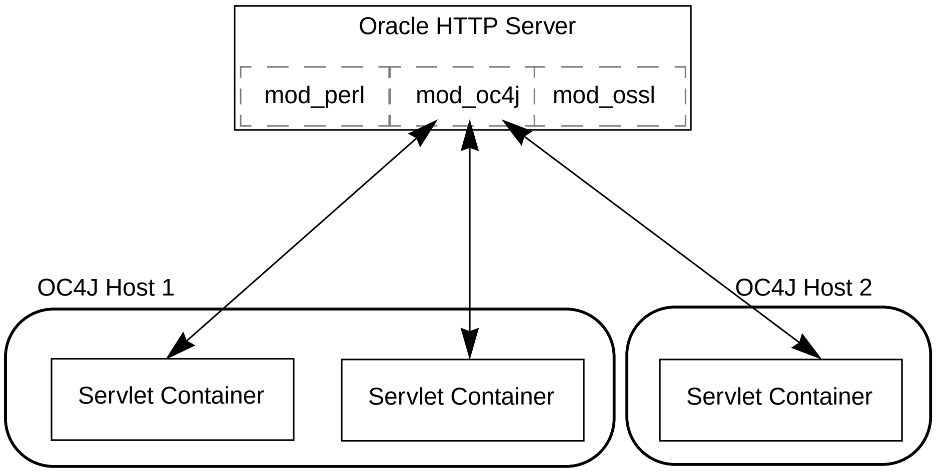
The `mod_oc4j` interacts with two components, Oracle Process Manager and Notification System (OPMN) and OC4J. The `mod_oc4j` interacts with OC4J by routing requests to it. OPMN starts Oracle HTTP Server, which starts `mod_oc4j`, and starts all OC4J processes. OPMN monitors each process that it starts and periodically verifies that each process is reachable. If a process dies, or becomes unreachable, OPMN will restart that process. In addition, OPMN communicates OC4J process status to `mod_oc4j` so that `mod_oc4j` knows when OC4J processes are started and stopped. The `mod_oc4j` uses this information to maintain an internal OC4J process table for rapid request routing.

> **See Also:** *Oracle9iAS Containers for J2EE User's Guide* in the Oracle9*i*AS Documentation Library

Figure 2–2 illustrates a one-to-many configuration. A one-to-many configuration, consists of one Oracle HTTP Server listner and multiple OC4J instances. In Figure 2–2, a single OC4J instance is communicating with two OC4J hosts. OC4J Host 1 is running two servlet containers, and OC4J Host 2 is running one servlet container. Three connections are open between the servlet containers and a single `mod_oc4j` in the OC4J instance.

A servlet container provides the runtime environment to execute servlets implementing the Servlet 2.3 Application Programming Interface (API) specifications. It runs in a JVM process in the same or different host as the OC4J. Each JVM has one servlet container, and the number of servlet containers is not proportional to the number of Web servers (`mod_oc4j` modules). One `mod_oc4j` can work with more than one servlet container and vice versa. Or, multiple `mod_oc4j` modules can work with multiple servlet containers.

Figure 2–2 AOC4J Architecture (one-to-many example)



Single Host Configuration

When a servlet container is located on the same machine as the Web server, you can set up the `mod_oc4j` module to start or stop the servlet container 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_oc4j` can also perform failover by checking JVM status regularly and starting another JVM if the first one becomes unavailable.

Life cycle

JWeb classes and OC4J applications 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()`.

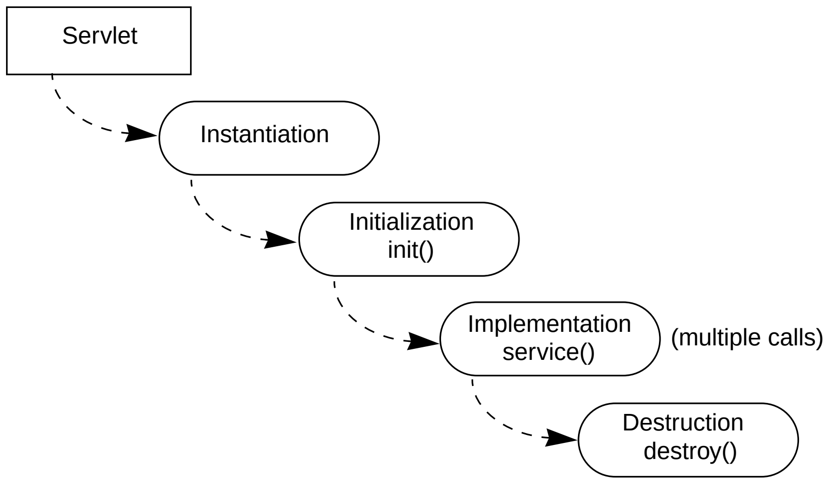
> **See Also:** Information on Java Virtual Machines from `http://java.sun.com/docs`

OC4J Life Cycle

In OC4J, Servlet life cycle is in compliance with Servlet 2.3 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 illustrates the servlet life cycle.

Figure 2–3 Servlet Life Cycle



Threading

The JWeb cartridge and OC4J servlet container 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. If the stateless parameter is set to `true`, then a cartridge instance is shared by more than one client. If the stateless perimeter is set to `false`, then 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.

OC4J Threading

The OC4J servlet container is multi-threaded by default. The OC4J servlet container 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 OC4J, in accordance with Servlet 2.3 specifications, only programmable sessions are available. Consequently, if you are migrating a JWeb application that was session-enabled by means other than code, you must implement the session mechanism programmatically using the servlet session API. See "Session Control" on page 2-9.

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 Oracle9*i*AS. If you would like to embed dynamic information in HTML pages (scripting), consider using JSP in Oracle9*i*AS.

> **See Also:** *Oracle9iAS Containers for J2EE Support for JavaServer Pages Reference* in the Oracle9*i*AS Documentation Library

Migration Strategies

OC4J provides complete support for J2EE 1.2, as well as support for major J2EE 1.3, such as complete Servlet 2.3, partial EJB 2.0 (message-driven beans), complete Jaas and JCA support. If you have JWeb or JServlet applications deployed on Oracle Application Server 4.x and wish to migrate to Oracle9*i*AS, you must modify your JWeb or JServlet applications to comply with applicable specifications for OC4J.

Comparison of Compliance Standards

Table 2–1 presents the comparison of compliance standards between the JWeb and JServlet cartridges in Oracle Application Server, and OC4J in Oracle9*i*AS. When migrating JWeb or JServlet from Oracle Application Server to OC4J Servlet in Oracle9*i*AS, you must modify the code to comply with Servlet 2.3 specifications.

Table 2–1 Comparison of Compliance Standards for JWeb, JServlet, and OC4J

Standard Complied	JWeb	JServlet	OC4J
Servlet Specifications	NA	2.1	2.3
JSP Specifications	NA	NA	1.1*

* JSP 1.2 is being completed when preparing this document. Please consult your product to verify actual specifications conplied.

See Also: `http://java.sun.com` for more information regarding Servlet specifications**:**

Key JWeb & Servlet Methods

In order to migrate, you must understand and use the following key methods:

JWeb—contains a java class with a `main()` method, also known as JWeb Cartridge. The infrastructure of JWeb maps a URL to this method.

Servlet—contains a java class that includes a few `doGet()` and `doPut()` methods, specified by Sun Microsystems Inc., which map to a URL.

Migration Approach

As a primary migration approach, you can call the `main()` method of the JWeb Cartridge in the corresponding `doGet()` Servlet method.

Specifically, you must focus on the following aspects:

- **Logging APIs**—Oracle9iAS does not support the Oracle Application Server logging APIs. Instead, it uses the Servlet logging APIs. Therefore, you must modify your code in JWeb cartridge to reflect the changes in logging APIs.
- **Utility APIs**—Use JSP to write your utility APIs. Currently, `oracle.html.*.` package is not available.

  See Also: `http://jakarta.apache.org/ecs/index.html` for more information regarding `html.*.` packages

- **WRB Calls**—You must use the standard servlet APIs to write your security code since Oracle9*i*AS does not support most WRB APIs. For example, you can use methods `getClientCertificate()`and `getLogger()`.
- **Session**—see "Session Control" on page 2-9.
- **Application Thread**—see "Application Threads" on page 2-10.
- **Logging**—see "Logging" on page 2-10.

Code Modifications for JWeb Applications

To migrate JWeb applications to OC4J, you must 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 OC4J, as per the Servlet 2.3 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 OC4J, there is no support for configurable sessions. Therefore, you must 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 be stored subsequently and retrieved, for example, by the `setAttribute(name, who)` and `getAttribute(name)` methods of `javax.servlet.http.HttpSession`, respectively.

```
session.setAttribute(“List”, new Vector());
Vector list = (Vector) session.getAttribute(“List”);
```

Note: Do not use static data members to maintain session state in OC4J (although this is a common practice in JWeb). Instead, use the Servlet session API. The latter allows the Servlet container to use memory more efficiently.

Session Timeout

The default session timeout for an OC4J container can be specified in the `session.config` element in the XML deployment descriptors. A programmer can use the `getMaxInactiveInernal` method in the `HTTPSession` interface. Use `setMaxInactiveInternal` method to set the time-out value for a container.

The JWeb session time-out callback is not available in OC4J.

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 OC4J, you only need standard Java thread management to manage application threads (the `java.lang.Runnable` interface is used). For both JWeb and OC4J, using application threads is not recommended because multi-threaded applications limit the effectiveness of the load balancer.

Logging

In Oracle Application Server, JWeb applications log messages using the 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 Oracle9*i*AS, OC4J generates diagnostic messages associated with Servlet logging APIs. These logging files are located at:

ORACLE_HOME/j2ee/home/log/*digit digit_island-name*/server.log

Note: OC4J does not have log levels.

JWeb Toolkit Packages (JWeb API)

Oracle Application Server includes a JWeb toolkit containing proprietary Java packages. If you used any of those packages in JWeb applications that you are migrating to Oracle9*i*AS, you must modify the code to use Servlet 2.3 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.

Table 2–2 through Table 2–8 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 2–7, "JWeb Equivalents for Javax.servlet.ServletOutputStream Class Methods"
- Table 2–8, "JWeb Equivalents for javax.servlet.ServletInputStream Class Method"

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(string)
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] return a hashtable of header names and values
[2] return an enumeration of header names

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()

Table 2–3 JWeb Equivalents for javax.servlet.ServletRequest Class Methods

JWeb Method	Servlet Method
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.getPreferredAcceptCharset()	getCharacterEncoding()
oracle.owas.wrb.services.http.HTTP.getURLParameter(name)	getParameter(string)
oracle.owas.wrb.services.http.HTTP.getURLParameters(name)	getParameterValues(string)

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)

Table 2–7 JWeb Equivalents for Javax.servlet.ServletOutputStream Class Methods

JWeb Method	Servlet Method
oracle.html.HtmlStream.print()	javax.servlet.ServletOutputStream.print()
oracle.html.HtmlStream.println()	avax.servlet.ServletOutputStream.println()

Table 2–8 JWeb Equivalents for javax.servlet.ServletInputStream Class Method

JWeb Method	Servlet Method
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 in Oracle9*i*AS, and discusses considerations for migrating cartridges to the Oracle9*i*AS infrastructure. The topics include:

Cartridge Types and Corresponding Oracle9*i*AS Modules

Table 3–1 presents the equivalent Oracle Application Server cartridge types and their Oracle9*i*AS components:

Table 3–1 Cartridge Types

Oracle Application Server Cartridge Type	**Oracle9*i*AS Equivalent**
PL/SQL	`mod_plsql`
Perl	`mod_perl`
LiveHTML	Apache SSI, and JSP
CWeb	Custom Apache Modules, FastCGI, CGI, Java JNI and PL/SQL Callouts

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

Note: Oracle Application Server uses Perl version 5.004_01, while Oracle9*i*AS uses the uses Perl version 5.6.1. When code modification is required, please use the appropriate Perl version.

PL/SQL Migration

You can migrate Oracle Application Server PL/SQL Cartridge applications to Oracle9*i*AS `mod_plsql`. Both `mod_plsql` and PL/SQL Cartridge applications provide similar support for building and deploying PL/SQL-based applications on the Web.

The `mod_plsql` runs as an Oracle HTTP Server module. It delegates the servicing of HTTP requests to PL/SQL programs, which execute their logic inside Oracle databases.

If you are planning to migrate PL/SQL applications from Oracle Application Server to Oracle9*i*AS, you should read *Oracle9i Application Server mod_plsql User's Guide* in the Oracle9*i*AS Documentation Library to be familiar with the features in this module.

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

File Upload and Download

Table 3–2 summarizes the file upload and download features supported by Oracle Application Server and Oracle9*i*AS.

Table 3–2 File Upload and Download Features Comparison

File Upload/Download Features	Oracle Application Server Support	Oracle9*i*AS 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 or download	Yes	No
Upload multiple files per form submission	Yes	Yes

Note: All Oracle Application Server features are supported in Oracle9*i*AS, except file `compression/decompression`. Users with compressed uploaded files in Oracle Application Server do not need to decompress their files manually. These files will be automatically decompressed and uploaded in an uncompressed format into the Oracle9*i*AS Document Table (see Table 3–3). This process is performed by the `oas2ias` file migration tool (see "Using the oas2ias Tool" on page 3-5).

See Also: *Oracle9i Application Server mod_plsql User's Guide* in the Oracle9*i*AS Documentation Library

Uploaded File Document Format

Oracle Application Server PL/SQL Cartridge and Oracle9*i*AS `mod_plsql` both support uploading files. However, they use different document table schemas. Users with uploaded files on Oracle Application Server who wish to migrate to Oracle9*i*AS must 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*AS tables while maintaining the uploaded content and the content description.
- Deflating compressed content in Oracle Application Server before migrating to Oracle9*i*AS. This version of Oracle9*i*AS does not support compression/decompression for uploaded files (see the previous section for further details).

The `oas2ias` tool reads all the rows from the `OWS_CONTENT` table and populates the content and attributes to a document table you specify.

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

Table 3–3 Derived Column Values

Column in Oracle9*i* 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`

Table 3–3 Derived Column Values

Column in Oracle9*i* Application Server Document Table	Oracle Application Server table.column Value
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*AS document table. The tool will also ensure that the data loaded into the Oracle9*i*AS 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*AS.

Using the oas2ias Tool

You only need to run the oas2ias tool once to convert all Oracle Application Server files to Oracle9*i*AS format with the following steps:

1. Make sure you have a current backup of all Oracle Application Server uploaded files.
2. Create the document table for Oracle9*i*AS. You can create this as 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*AS Release 2 (9.0.2) or later

- Oracle database version 9.x or later
- *ORACLE_HOME* is set to the Oracle9*i*AS Oracle Home directory.
- For Windows, the system path contains *ORACLE_HOME*\bin
- For UNIX, the PATH environment variable contains *ORACLE_HOME*/bin
- For UNIX, 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 Oracle9*i*AS database (where the Oracle9*i* Application Server user schema with the my_doc_table table exists). Store the aliases in the following directory:

   - (UNIX) *ORACLE_HOME*/network/admin/tnsnames.ora
   - (Windows) *ORACLE_HOME*\network\admin\tnsnames.ora

   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*AS 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 Oracle9*i*AS document table created in step 2
ias_password	password for ias_user_name
ias_connstr	connect string for the mod_plsql database

Parameter	Description
ias_doc_table	name of the Oracle9*i*AS 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>: db
iAS database user: oracle
iAS database user's password: welcome
iAS database connect string <ENTER if local database>: db
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*AS document table and the files are now available for access using Oracle9*i*AS mod_plsql.

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 mod_plsql also supports custom authentication.

> **See Also:** *Oracle9i Application Server mod_plsql User's Guide* in the Oracle9*i*AS Documentation Library

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 behavior depending on how many parameters a form passes to the procedure.

Both Oracle Application Server and Oracle9*i*AS support flexible parameter passing. To use flexible parameter passing in the `mod_plsql`, 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 `mod_plsql` procedure will be:

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

See Also: *Oracle9i Application Server mod_plsql User's Guide* in the Oracle9*i*AS Documentation Library

Positional Parameter Passing

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

See Also: *Oracle9i Application Server mod_plsql User's Guide* in the Oracle9*i*AS Documentation Library

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.

Oracle9*i*AS does not support this feature. You must 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 you can migrate them to Oracle9*i*AS.

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_01.   The interpreter is built as the following:

- (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 input/output (for example, 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 input/output.

- Supports error logging.
- Supports performance instrumentation.

You can run your Perl scripts developed for Oracle Application Server under the CGI environment in Oracle9*i*AS CGI environment, as well, after modifying the interpreter line of your Perl scripts. You may also modify your Perl scripts for Perl cartridge in Oracle Application Server in order to run under Oracle9*i*AS.

Migrating Perl Cartridge Scripts

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

The Oracle9*i*AS Perl Environment

The Oracle9*i*AS Perl environment is based on `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: *Oracle9i* Application Server *mod_plsql User's Guide* documentation in the Oracle9*i*AS Documentation Library

Perl Modules

Table 3–4 presents comparisons of the third party Perl modules associated with both Oracle Application Server and Oracle9*i*AS. In order to migrate applications that use these modules from Oracle Application Server to Oracle9*i*AS, you must acquire these modules and install them. The files are available from:

`http://www.cpan.org`

Table 3–4 Comparison of Third Party Perl Modules

Perl Module	Version in Oracle Application Server	Version in Oracle9*i*AS
DBI	0.79	1.20
DBD::Oracle	0.44	1.12
LWP or libwww-perl	5.08	5.53_94
CGI	2.36	2.752

Table 3–4 Comparison of Third Party Perl Modules

Perl Module	**Version in Oracle Application Server**	**Version in Oracle9*i*AS**
MD5	1.7	2.14
IO	1.15	1.20
NET	1.0502	1.0703
Data-Dumper	2.07	NA
Apache DBI	NA	0.88
Devel::Symdump	NA	2.01
Digest::HMAC	NA	1.01
Digest::MD2	NA	2.00
Digest::SHA1	NA	2.00
HTML::Parser	NA	3.25
MIME::Base64	NA	2.12
PlRPC	NA	0.2015
Storable	NA	1.0.12
Net::Daemon	NA	0.35
Time::HiRes	NA	1.20
URI	NA	1.15

Variations from Oracle Application Server Perl Cartridge

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

Namespace Collision

Both Oracle Application Server and Oracle9*i*AS cache compiled Perl scripts. 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*AS 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*AS form the package name differently. Oracle Application Server cannot cache subroutines with the same name. Oracle9*i*AS 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*AS.

> **See Also:** `http://cgi-lib.stanford.edu/cgi-lib` for more information on `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.

The corresponding pre-load script for Oracle9*i*AS is `startup.pl`.

> **See Also:** `http://perl.apache.org` for more information on `mod_starup.pl`

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, or by using Perl for scripting. If you are migrating LiveHTML applications to Oracle9*i*AS, you must migrate LiveHTML SSI to Apache SSI. Currently the only equivalent to LiveHTML embedded scripts in Oracle9*i*AS is JSP.

SSI

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
flastmod	flastmod
include	include
printenv	not available
set	not available
not available	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 OC4J by default.

In addition to the elements shown in the table above, Apache SSI also includes variable substitution and flow control elements.

Scripts

In Oracle Application Server, you can use the LiveHTML cartridge to embed Perl scripts in HTML files. There is no equivalent functionality in Oracle9*i*AS. However, you have the following choices to do so.

1. Keep the logic in Perl and use `mod_perl`—for example, you can change the HTML piece to `printf()`.
2. Keep the HTML, but change the programming language to PL/SQL.
3. Download from the Web for tools that allow using Perl as a scripting language with HTML, for example at `http://perl.apache.org/#appservers`.

Note: The tools run on top of `mod_perl`. Therefore, this migration approach is the easiest, comparing to other three approaches listed in this section.

4. Keep the HTML, but change the programming language to Java, for example JSP—Oracle9*i*AS complies with JSP 1.1 specifications. To migrate you LiveHTML application to Oracle9*i*AS, you must do the following:
   a. Migrate from the LiveHTML application model to the JSP application model.
   b. Migrate LiveHTML tags to JSP tags.
   c. Rewrite the Perl code as Java code.

Note: 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 JSP.

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

Note: Oracle9*i*AS does not provide WRB APIs.

CWeb Migration

In Oracle Application Server, you can use the CWeb Cartridge to:

- create custom cartridges
- develop applications that other cartridges invoke

The migration paths from Oracle Application Server CWeb Cartridges to Oracle9*i*AS include:

- Using FastCGI
- creating a custom Oracle9*i*AS module

Using FastCGI

CWeb cartridge is essentially a `.DLL` or a `.so` library. You can integrate into the Oracle Application Server environment by specifying the entry point of this library in an administration page and map it to a Web URL. The Oracle Application Server infrastructure invokes the entry point of the library (CWeb cartridge) when a browser requests that URL. In addition, the CWEB cartridge makes several API from the WRB infrastructure available to access the client information, and other environment information.

CGI is a standard supported by all Web servers, including Oracle9*i*AS. When a URL that maps to the "CGI program" is accessed, the Web server will start that program and return its results to the browser.

Therefore, one simple way to migrate CWeb is to write a simple C program that invokes the entry point of the CWeb cartridge during the start-up.

The WRB API and other Oracle Application Server infrastructure dependencies will, of course, not be available in the new Oracle9*i*AS environment. If these WRB API or capabilities were used, the CWeb cartridge must be modified to use alternative API.

From an infrastructure standpoint, the CWeb cartridges were load balanced. New instances were not started on each request.

However, CGI causes the invocation of a new program on each request and does not support program reuse. Beginning Oracle9*i*AS v1.0.2.2, this problem is solved with the introduction of FastCGI.

FastCGI is an "overloaded" term referring to the specifications, protocol, API, and also the implementation. In summary, it spawns a separate process and keeps it alive and independent of the life-style of the requests. FastCGI programs must conform to certain standards for starting point and events to listen to, which is similar to a Java Servlet specification. Their life-cycle can, then, be controlled by the infrastructure.

Migrating a CWeb cartridge is similar to writing a FastCGI program, which conforms to the specifications and in turn calls the entry point of the CWeb cartridge. For FastCGI examples, refer to `http://www.fastcgi.com`.

Note: The same limitations of WRB API mentioned earlier still apply, since the Oracle9*i*AS does not provide WRB APIs.

Creating a Custom Oracle9*i*AS Module

If you used CWeb to create custom cartridges you can also consider creating a custom Oracle9*i*AS module.

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

- CGI scripts: stand-alone C programs generating Web content with `println` statements.
- Java JNI: Java Servlets or JSP that call CWeb routines from OC4J
- PL/SQL callouts: PL/SQL applications that call CWeb routines from Oracle Database Cache or Oracle9*i*.

Note: Oracle9*i*AS does not provide WRB and CWeb APIs.

---

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 OC4J in Oracle9*i*AS. The topics include:

- Migrating EJBs to OC4J
- Migrating ECO/Java to OC4J
- Migrating JCORBA to OC4J

Migrating EJBs to OC4J

To migrate EJBs from Oracle Application Server 4.x to OC4J, you must modify code in the following areas:

- Deployment Descriptors
- Client Code
- Logging (Server Code) (if applicable)

Note: Oracle Application Server EJB does not comply to EJB standards, while Oracle9*i*AS EJB complies with complete EJB 1.2 and partial EJB 2.0 specifications. Please modify your code accordingly during the migration.

The following sections describe these changes.

Deployment Descriptors

OC4J conforms to XML file configuration that complies to J2EE 1.2 specifications.

See Also: *Oracle9iAS Containers for J2EE Enterprise JavaBeans Developer's Guide and Reference* in the Oracle9*i*AS 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 following properties:

- `javax.naming.Context.URL_PKG_PREFIXES`
- `javax.naming.Context.SECURITY_AUTHORIZATION`
- `javax.naming.Context.SECURITY_PRINCIPAL`
- `javax.naming.Context.SECURITY_CREDENTIALS`

See Also: Chapter 3, "Advanced Configuration, Development, and Deployment" in *Oracle9iAS Containers for J2EE User's Guide* in the Oracle9*i*AS documentation library

You must also change the URL that accesses your EJB home to the OC4J:

```
ORMI://<host>:<port>/<path>/<bean>
```

For example:

```
ORMI://myhost:2481/test/myBean
ORMI://host/port/est/bean
```

Logging (Server Code)

If application logging was done in Oracle Application Server, remove all references to `oracle.oas.ejb.Logger` from your EJB code.

Migrating ECO/Java to OC4J

When migrating ECO for Java (ECO / Java) in Oracle Application Server to OC4J in Oracle9*i*AS, you must change server code described in this section, as well as to change deployment descriptors and client code described in the previous section for EJB migration.

To make your ECO / Java components compatible with OC4J, you must modify the implementation file, the remote interface file, the home interface file, and deployment descriptors.

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 created 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.
7. OC4J uses XML files for deployment, you have to create appropriate deployment files.

Migrating JCORBA to OC4J

Oracle Application Server versions 4.0.6 and 4.0.7 provided a component model, Java CORBA Objects (JCO), which is a precursor to the ECO / Java model. Oracle9*i*AS does not support CORBA objects. You must recode your CORBA objects as EJBs. This section discusses migration from JCO in Oracle Application Server to OC4J in Oracle9*i*AS.

To migrate to OC4J, you must modify the server and client code as discussed in this section. 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 must to create a home interface, as defined in the EJB specification. The following is an example.

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

Object Implementation

Complete the following steps to migrate the implementation class:

1. Change import `oracle.oas.jco.*` to import `javax.ejb.*`.

2. Check that the class implements `javax.ejb.SessionBean`, or `javax.ejb.EntityBean`.

> **Note:** The JCORBA Lifecycle is not supported within OC4J. 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 `java.rmi.RemoteException`.
7. Change any `ObjectManager` type to `SessionContext` type. Table 4–1 maps the methods in the ObjectManager class to methods in the SessionContext class.

Table 4–1 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