Oracle9*i* Application Server

Best Practices

Release 2 (9.0.3)

Part No. B10578-02

August 2003

ORACLE®

Oracle9*i* Application Server Best Practices, Release 2 (9.0.3)

Part No. B10578-02

Contributing Author: Jesse Anton, Jerry Bortvedt, Alice Chan, Christine Chan, Lee Cooper, Helen Grembowicz, Kurt Heiss, Pavana Jain, Valerie Kane, John Lang, Bill Lankenau, Bruce Lowenthal, Kuassi Mensah, Mukul Paithane, Debu Panda, Jason Pepper, Shiva Prasad, Kalle Radage, Venkat Ravipati, Blaise Ribet, Jeffrey Schafer, Daniel Shih, Jerry Silver, Deborah Steiner, Harry K. Wong

Contents

Send Us Your Comments

Oracle9*i* Application Server Best Practices, Release 2 (9.0.3)

Part No. B10578-02

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 about this manual?

If you find any errors or have any other suggestions for improvement, please indicate the title and part number of the document, 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: 650-506-7365 Attn: Oracle9*i* Application Server Documentation Manager
- Postal service:
  Oracle Corporation
  Oracle9*i* Application Server Documentation
  500 Oracle Parkway, M/S 1op6
  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 document describes the best practices for Oracle9*i* Application Server (Oracle9*i*AS).

This preface contains these topics:

Audience

This document is intended for all users of Oracle9*i*AS: application designers, developers, deployers, and administrators. It is assumed that the reader has sufficient knowledge about J2EE and relevant Web technologies and terms.

The reader is also assumed to be familiar with basicOracle9*i*AS terminology. This document does not explain the terms to the beginner, but does cover the nuances of different options that a feature may provide.

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.

Organization

This document contains:

Chapter 1, "Introduction"
This chapter describes the Oracle9*i*AS Best Practices document content.

Chapter 2, "Java Language Best Practices"
This chapter describes best practices for Java language.

Chapter 3, "J2EE Best Practices"
This chapter describes best practices for J2EE language.

Chapter 4, "Oracle9iAS Framework Best Practices"
This chapter describes best practices for Oracle9*i*AS framework.

Chapter 5, "Oracle9iAS Web Cache Best Practices"
This chapter describes best practices for Oracle9*i*AS Web Cache.

Chapter 6, "Oracle HTTP Server Best Practices"
This chapter describes best practices for Oracle HTTP Server.

Chapter 7, "Oracle9iAS Portal Best Practices"
This chapter describes best practices for Oracle9*i*AS Portal.

Chapter 8, "Oracle9iAS Wireless Best Practices"
This chapter describes best practices for Oracle9*i*AS Wireless.

Chapter 9, "Security Best Practices"
This chapter describes best practices for security.

Chapter 10, "Oracle Enterprise Manager Best Practices"
This chapter describes best practices for Oracle Enterprise Manager.

Chapter 11, "Installation Best Practices"
This chapter describes best practices for installation.

Chapter 12, "Deployment Best Practices"

This chapter describes best practices for deployment.

Chapter 13, "Miscellaneous Best Practices"

This chapter describes best practices for miscellaneous.

Appendix A, "Oracle9iAS Web Cache Best Practices Appendix"

This appendix describes additional best practices for Oracle9*i*AS Web Cache.

Related Documentation

For more information, see these Oracle resources:

- Oracle9*i*AS Documentation Library CD-ROM
- Oracle9*i*AS Platform Specific Documentation
- Oracle9*i* Application Server Install Frequently Asked Questions at:

  `http://otn.oracle.com/products/ias/install-faq.html`

Printed documentation is available for sale in the Oracle Store at

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

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/membership/`

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/documentation/content.html`

Conventions

The following conventions are also used in this manual:

Convention	Meaning
⋮	Vertical ellipsis points in an example mean that information not directly related to the example has been omitted.
. . .	Horizontal ellipsis points in statements or commands mean that parts of the statement or command not directly related to the example have been omitted
boldface text	Boldface type in text indicates a term defined in the text, the glossary, or in both locations.
italic text	Italicized text indicates placeholders or variables for which you must supply particular values.
[ ]	Brackets enclose optional clauses from which you can choose one or none.

1 Introduction

This document is a collection of common practices regarding development and deployment of Oracle9*i*AS. It covers common mistakes, product usage scenarios, and also provides a review checklist for different phases of Oracle9*i*AS deployment. Some of the practices may be generic to Java 2 Enterprise Edition (J2EE) and not specific to Oracle9*i*AS.

The document introduces the J2EE and Web Cache install type of Oracle9*i*AS, and then discusses the practices associated with each Oracle9*i*AS component. The practices are then bound together with some complete deployment examples.

1.1 About Best Practices

A best practice is, in general, a recommendation or a common practice on how to perform certain tasks and actions. This recommendation may involve a combination of tools and manual processes to achieve a desired result.

The best practices in this document are deliberately kept brief. They do not include the detailed steps. Those are covered in other documentation library books.

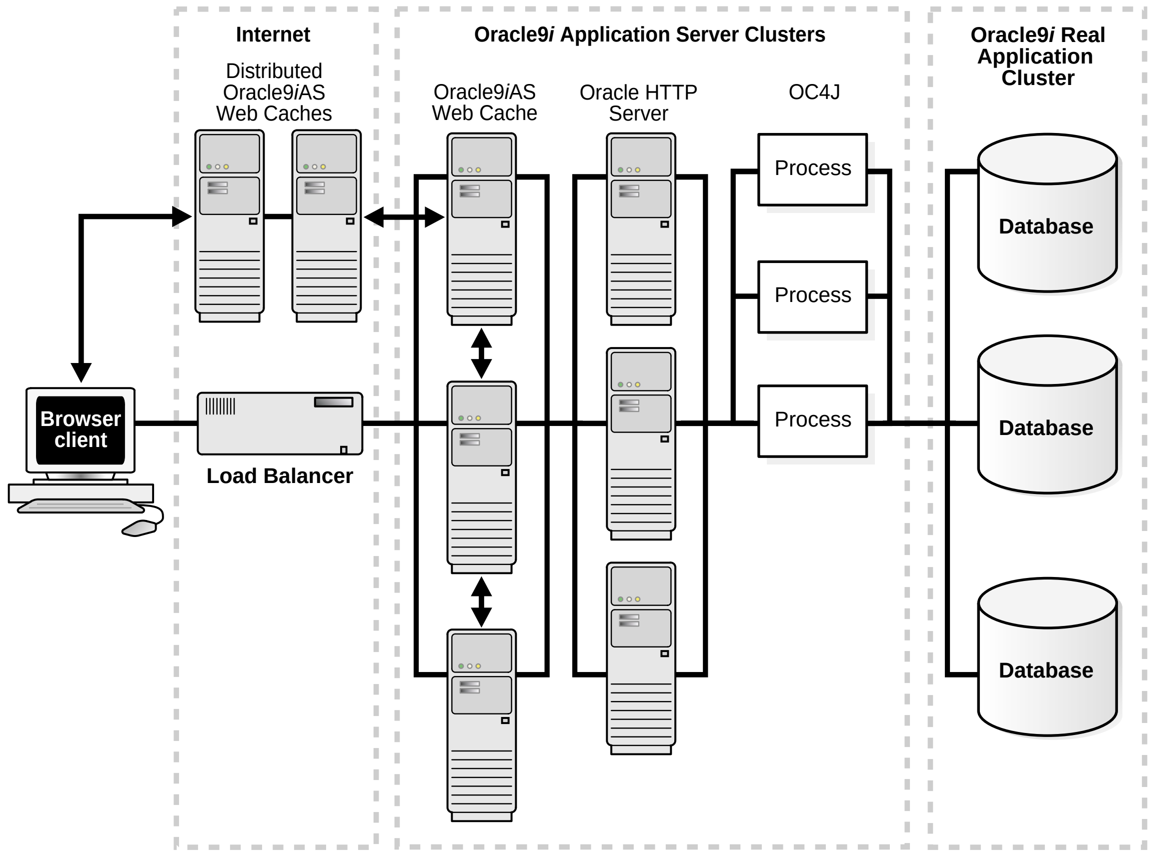
1.2 About Oracle9*i*AS J2EE and Web Cache Install Type

Oracle9*i*AS Release 2 can be installed in different ways– J2EE and Web Cache install type offers the most basic install. In this paper, the term Oracle9*i*AS is used to refer to this installation type. This book will discuss the following components:

- Oracle9*i*AS Web Cache: This is the first component of Oracle9*i*AS to receive requests. For both static and dynamic requests, it can cache the results, thus reducing the workload of the Web server machines behind it.
- Oracle HTTP Server: This is the Oracle9*i*AS component that services HTTP requests. It responds to requests forwarded to it by Oracle9*i*AS Web Cache. The Oracle HTTP Server sub-system is comprised of a Web server (based on Apache), a Perl execution environment, and a PL/SQL and OC4J routing system. Unless mentioned otherwise, a reference to Oracle HTTP Server usually is a reference to the Web server piece within Oracle HTTP Server.
- Oracle9*i*AS Containers for J2EE (OC4J): This is the J2EE-compliant container in Oracle9*i*AS. It provides clustering capabilities for the J2EE components –servlets, Java Server Pages (JSP), and Enterprise Java Beans (EJB). It also contains other mechanisms, such as Java Object Cache, which provides distributed caching capabilities.
- Oracle Enterprise Manager: Oracle Enterprise Manager is a Web-based administration tool for Oracle9*i*AS Release 2.
- Infrastructure Repository: This is a database or file-based repository for Oracle9*i*AS Release 2. It is used to persist deployment and configuration information for other Oracle9*i*AS components.
- Clustering Infrastructure Components: There are some components (Oracle Process Manager and Notification Service, mod_oc4j, and Distributed Configuration Manager) that work in tandem to provide significant clustering capabilities in Oracle9*i*AS Release 2.

Figure 1–1 below displays the components deployed in a clustered scenario.

Figure 1–1 Clustered Components



For detailed information on Oracle9*i*AS components and features, refer to the product documentation.

1.3 Audience

This document is intended for all users of Oracle9*i*AS: application designers, developers, deployers, and administrators. It is assumed that the reader has sufficient knowledge about J2EE and relevant Web technologies and terms.

The reader is also assumed to be familiar with basic Oracle9*i*AS terminology. This document does not explain the terms to the beginner, but does cover the nuances of different options that a feature may provide.

See Also: *Oracle9i Application Server Concepts Guide*

1.4 Document Organization

This document as a whole focuses on Oracle9*i*AS Web Cache, Oracle HTTP Server, OC4J, J2EE programming, Oracle9*i*AS Portal, Oracle Internet Directory, and Oracle9*i*AS deployment. Other mature technologies like Perl and PLSQL have sufficient associated literature, and are not covered here. An application server running J2EE applications is still impacted by the basic practices in Java programming – specially the performance-related practices.

These are covered first, followed by J2EE practices: JSPs, servlets, and EJBs. We then cover practices on some of the frameworks Oracle9*i*AS provides, for example, BC4J and Java Object Cache.

Oracle9*i*AS Web Cache practices then elaborate the different manners in which Oracle9*i*AS Web Cache may be leveraged. The overall security practices then provide some tips to secure deployment.

This document also describes some deployment architectures with respect to placing different components along with firewalls.

1.5 References and Relation to Other Documents

Most of the performance related practices are covered in more detail in the Oracle9*i*AS Performance and Tuning Guide. The sections on servlets, BC4J, and data access rely heavily on this.

In the clustering sections, the following two white papers were referenced:

- Oracle9*i*AS: Scalability, Availability, and Manageability of J2EE and Web Clusters at:

  `http://otn.oracle.com/products/ias/ohs/collateral/r2/clusters.pdf`

- mod_oc4j: A Technical Overview at:

  `http://otn.oracle.com/products/ias/ohs/collateral/r2/mod_oc4j_wp.pdf`

Other references as appropriate to certain sections or practices are alluded to within the description of the best practice.

2

Java Language Best Practices

This chapter describes Java language best practices. The topics include:

2.1 Avoid or Minimize Synchronization

Many performance studies have shown a high performance cost in using synchronization in Java. Improper synchronization can also cause a deadlock, which can result in complete loss of service because the system usually has to be shut down and restarted. But performance overhead cost is not a sufficient reason to avoid synchronization completely. Failing to make sure your application is thread-safe in a multithreaded environment can cause data corruption, which can be much worse than losing performance. The following are some practices that you can consider to minimize the overhead:

- Synchronize Critical Sections Only
- Do Not Use the Same Lock on Objects That Are Not Manipulated Together
- Use Private Fields
- Use a Thread Safe Wrapper
- Use Immutable Objects
- Know Which Java Objects Already Have Synchronization Built-in
- Do Not Under-Synchronize

2.1.1 Synchronize Critical Sections Only

If only certain operations in the method must be synchronized, use a synchronized block with a mutex instead of synchronizing the entire method. For example:

```
private Object mutex = new Object();
    …
    private void doSomething()
    {
     // perform tasks that do not require synchronicity
        …
        synchronized (mutex)
            {
            …
            }
        …
    }
```

2.1.2 Do Not Use the Same Lock on Objects That Are Not Manipulated Together

Every Java object has a single lock associated with it. If unrelated operations within the class are forced to share the same lock, then they have to wait for the lock and must be executed one at a time. In this case, define a different mutex for each unrelated operation that requires synchronization.

Also, do not use the same lock to restrict access to objects that will never be shared by multiple threads. For example, using `Hashtables` to store objects that will never be accessed concurrently causes unnecessary synchronization overhead:

```
public class myClass
{
    private static myObject1 myObj1;
    private static mutex1 = new Object();
    private static myObject2 myObj2;
    private static mutex2 = new Object();
    …
    public static void updateObject1()
        {
            synchronized(mutex1)
            {
            // update myObj1 …
            }
        }
    public static void updateObject2()
    {
    synchronized(mutex2)
        {
        // update myObj2 …
        }
    }
…
}
```

2.1.3 Use Private Fields

Making fields private protects them from unsynchronized access. Controlling their access means these fields need to be synchronized only in the class's critical sections when they are being modified.

2.1.4 Use a Thread Safe Wrapper

Provide a thread-safe wrapper on objects that are not thread-safe. This is the approach used by the collection interfaces in JDK 1.2.

2.1.5 Use Immutable Objects

An immutable object is one whose state cannot be changed once it is created. Since there is no method that can change the state of any of the object's instance variables once the object is created, there is no need to synchronize on any of the object's methods.

This approach works well for objects, which are small and contain simple data types. The disadvantage is that whenever you need a modified object, a new object has to be created. This may result in creating a lot of small and short-lived objects that have to be garbage collected. One alternative when using an immutable object is to also create a mutable wrapper similar to the thread-safe wrapper.

An example is the `String` and `StringBuffer` class in Java. The `String` class is immutable while its companion class `StringBuffer` is not. This is part of the reason why many Java performance books recommend using `StringBuffer` instead of string concatenation.

See Also: Section 2.12, "Use Stringbuffer Instead of String Concatenation"

2.1.6 Know Which Java Objects Already Have Synchronization Built-in

Some Java objects (such as `Hashtable`, `Vector`, and `StringBuffer`) already have synchronization built into many of their APIs. They may not require additional synchronization.

2.1.7 Do Not Under-Synchronize

Some Java variables and operations are not atomic. If these variables or operations can be used by multiple threads, you must use synchronization to prevent data corruption. For example: (i) Java types long and double are comprised of eight bytes; any access to these fields must be synchronized. (ii) Operations such as ++ and –- must be synchronized because they represent a read and a write, not an atomic operation.

2.2 Monitor Synchronization

Java synchronization can cause a deadlock. The best way to avoid this problem is to avoid the use of Java synchronization. One of the most common uses of synchronization is to implement pooling of serially reusable objects. Often, you can simply add a serially reusable object to an existing pooled object. For example, you can add Java Database Connectivity (JDBC) and `Statement` object to the instance variables of a single thread model servlet, or you can use the Oracle JDBC connection pool rather than implement your own synchronized pool of connections and statements.

If you must use synchronization, you should either avoid deadlock, or detect it and break it. Both strategies require code changes. So, neither can be completely effective because some system code uses synchronization and cannot be changed by the application.

To prevent deadlock, simply number the objects that you must lock, and ensure that clients lock objects in the same order.

Proprietary JVM extensions may be available to help spot deadlocks without having to instrument code, but there are no standard JVM facilities for detecting deadlock.

2.3 Monitor and Fix Resource Leaks

One way to fix resource leaks is straightforward - a periodic restart. It provides good protection against slow resource leaks. But it is also important to spot applications that are draining resources too quickly, so that any software bugs can be fixed. Leaks that prevent continuous server operation for at least 24 hours must be fixed in the application code, not by application restart.

Common programming mistakes are:

- Not returning the resource to the pool (or not removing it from the pool) after handling an error.
- Relying on the garbage collector to invoke `finalize()` and free resources. Never rely on the garbage collector to manage any resource other than memory.
- Not discarding old object references which prevent recycling the memory occupied by the objects.

Monitoring resource usage should be a combination of code instrumentation and external monitoring utilities. With code instrumentation, calls to an application-provided interface, or calls to a system-provided interface like Oracle Dynamic Monitoring System (DMS), are inserted at key points in the application's resource usage lifecycle. Done correctly, this can give the most accurate picture of resource use. Unfortunately, the same programming errors that cause resource leaks are also likely to cause monitoring errors. That is, you may forget to release the resource, or forget to monitor the release of the resource.

Operating system commands like vmstat or ps in UNIX, provide process-level information such as the amount of memory allocated, the number and state of threads, or number of network connections. They can be used to detect a resource leak. Some commercially available development tools can also be used to find the leak.

In addition to compromising availability, resource leaks and overuse decrease performance.

See Also: Section 2.8, "Avoid Common Errors That Can Result In Memory Leaks"

2.4 Always Use a Finally Clause In Each Method to Cleanup

In Java, it is impossible to leave the `try` or `catch` blocks (even with a `throw` or `return` statement) without executing the finally block. If for any reason the instance variables cannot be cleaned, throw a catch-all exception that should cause the caller to discard its object reference to this now corrupt object. If, for any reason the static variables cannot be cleaned, throw an `InternalError` or equivalent that will ultimately result in restarting the now corrupt JVM.

2.5 Discard Objects That Throw Catch-All Exceptions

In many cases, these exceptions indicate that the internal state of the invoked object is corrupt, and that further invocations will also fail. Keep the object reference only if careful scrutiny of the exception shows it is benign, and further invocations on this object are likely to succeed.

Adopt a guilty unless proven innocent approach. For example, a `SQLException` thrown from an Oracle JDBC invocation could represent one of thousands of error conditions in the JDBC driver, the network, or the database server. Some of these errors (for example, subclass `SQLWarning`) are benign. Some `SQLExceptions` (for example, "ORA-3113: end of file on communication channel") definitely leave the JDBC object useless. Most `SQLExceptions` do not clearly specify what state the JDBC object is left in. The best approach is to enumerate the benign error codes that could occur frequently in your application and can definitely be retried, such as a unique key violation for user-supplied input data. If any other error code is found, discard the potentially corrupt object that threw the exception.

Discard all object references to the (potentially) corrupt object. Be sure to remove the corrupt object from all pools in order to prevent pools from being poisoned by corrupt objects. Do not invoke the corrupt object again – instantiate a brandnew object instead.

When you are sure that the corrupt objects have been discarded and that the catching object is not corrupt, throw a non catch-all exception so that the caller does not discard this object.

2.6 Design Transactions Usage Correctly

Transactions should not span client requests because this can tie up shared resources indefinitely.

Requests generally should not span more than one transaction, because a failure in mid-request could leave some transactions committed and others rolled back. If this requires application-level compensation to recover, then availability or data integrity may be compromised.

Transactions generally should not span more than one database, because distributed transactions lock shared resources longer, and failure recovery may require simultaneous availability and coordination of multiple databases.

Applications that require a single client request (for example, a confirm checkout request in a shopping cart application) to ultimately affect several databases (for example, credit card, fulfillment, shopping cart, and customer history databases) should perform the first step with one database, and in the same transaction queue a message in the first database addressed to the second database. The second database will perform the second step and queue the third step, and so on. This queued transaction chain will eventually complete automatically, or an administrator will see an undeliverable message and will have to manually compensate.

2.7 Put Business Logic In the Right Place

In general, you should not implement business logic in your client program. Instead, put validation and defaulting logic in your entity objects, and put client-callable methods in application modules, view objects, and view rows.

Working with application module methods allows the client program to encapsulate task-level custom code in a place that allows data-intensive operations to be done completely in the middle-tier without burdening the client.

Working with view object methods allows the client program to access the entire row collection for cross-row calculations and operations.

Working with view row methods allows the client program to operate on individual rows of data. There are three types of custom view row methods you may want to create:

- **Accessor methods**: The `oracle.jbo.Row` interface (which view rows implement) contains the methods `getAttribute()` and `setAttribute()`, but these methods are not typesafe. You can automatically generate custom typesafe accessors when you generate a custom view row class.
- **Delegators to entity methods**: By design, clients cannot directly access entity objects. If you want to expose an entity method to the client tier, you should create a delegator method in a view row.
- **Entity-independent calculations:** This is useful if the calculation uses attributes derived from multiple entity objects or from no entity objects.

2.8 Avoid Common Errors That Can Result In Memory Leaks

In Java, memory bugs often appear as performance problems, because memory leaks usually cause performance degradation. Because Java manages the memory automatically, developers do not control when and how garbage is collected. To avoid memory leaks, check your applications to make sure they:

- Release JDBC `ResultSet`, `Statement`, or connection.
- Release failures here are usually in error conditions. Use a `finally` block to make sure these objects are released appropriately.
- Release instance or resource objects that are stored in static tables.

Perform clean up on serially reusable objects.

An example is appending error messages to a `Vector` defined in a serially reusable object. The application never cleaned the `Vector` before it was given to the next user. As the object was reused over and over again, error messages accumulated, causing a memory leak that was difficult to track down.

See Also: Section 2.3, "Monitor and Fix Resource Leaks"

2.9 Avoid Creating Objects or Performing Operations That May Not Be Used

This mistake occurs most commonly in tracing or logging code that has a flag to turn the operation on or off during runtime. Some of this code goes to great lengths creating and formatting output without checking the flag first, creating many objects that are never used when the flag is off. This mistake can be quite expensive, because tracing and logging usually involves many `String` objects and operations to translate the message or even access to the database to retrieve the full text of the message. Large numbers of debug or trace statements in the code make matters worse.

2.10 Replace Hashtable and Vector With Hashmap, ArrayList, or LinkedList If Possible

The `Hashtable` and `Vector` classes in Java are very powerful, because they provide rich functions. Unfortunately, they can also be easily misused. Since these classes are heavily synchronized even for read operations, they can present some challenging problems in performance tuning. Hence, the recommendations are:

- Use an Array Instead of an ArrayList If the Size Can Be Fixed
- Use an ArrayList or LinkedList To Hold a List of Objects In a Particular Sequence
- Use HashMap or TreeMap To Hold Associated Pairs of Objects
- Replace Hashtable, Vector, and Stack
- Avoid Using String As the Hash Key (If Using JDK Prior to 1.2.2)

2.10.1 Use an Array Instead of an ArrayList If the Size Can Be Fixed

If you can determine the number of elements, use an `Array` instead of an `ArrayList`, because it is much faster. An `Array` also provides type checking, so there is no need to cast the result when looking up an object.

2.10.2 Use an ArrayList or LinkedList To Hold a List of Objects In a Particular Sequence

A `List` holds a sequence of objects in a particular order based on some numerical indexes. It will be automatically resized. In general, use an `ArrayList` if there are many random accesses. Use a `LinkedList` if there are many insertions and deletions in the middle of the list.

2.10.3 Use HashMap or TreeMap To Hold Associated Pairs of Objects

A Map is an associative array, which associates any one object with another object. Use a `HashMap` if the objects do not need to be stored in sorted order. Use `TreeMap` if the objects are to be in sorted order. Since a `TreeMap` has to keep the objects in order, it is usually slower than a `HashMap`.

2.10.4 Replace Hashtable, Vector, and Stack

- Replace a `Vector` with an `ArrayList` or a `LinkedList`.
- Replace a `Stack` with a `LinkedList`.
- Replace a `Hashtable` with a `HashMap` or a `TreeMap`.

`Vector`, `Stack`, and `Hashtable` are synchronized-views of `List` and Map. For example, you can create the equivalent of a `Hashtable` using:

```
private Map hashtable = Collections.synchronizedMap (new HashMap());
```

However, bear in mind that even though methods in these synchronized-views are thread-safe, iterations through these views are not safe. Therefore, they must be protected by a synchronized block.

2.10.5 Avoid Using String As the Hash Key (If Using JDK Prior to 1.2.2)

In Java's `HashMap` or `TreeMap` implementation, the `hashCode()` method on the key is invoked every time the key is accessed. If the hash key is a `String`, each access to the key will invoke the `hashCode()` and the `equals()` methods in the `String` class. Prior to JDK release 1.2.2, the `hashcode()` method in the `String` class did not cache the integer value of the `String` in an int variable; it had to scan each character in the `String` object each time. Such an operation can be very expensive. In fact, the longer the length of the `String`, the slower the `hashCode()` method.

2.11 Reuse Objects Instead of Creating New Ones If Possible

Object creation is an expensive operation in Java, with impact on both performance and memory consumption. The cost varies depending on the amount of initialization that needs to be performed when the object is to be created. Here are ways to minimize excess object creation and garbage collection overhead:

- Use a Pool to Share Resource Objects
- Recycle Objects
- Use Lazy Initialization to Defer Creating the Object Until You Need It.

2.11.1 Use a Pool to Share Resource Objects

Examples of resource objects are threads, JDBC connections, sockets, and complex user-defined objects. They are expensive to create, and pooling them reduces the overhead of repetitively creating and destroying them. On the down side, using a pool means you must implement the code to manage it and pay the overhead of synchronization when you get or remove objects from the pool. But the overall performance gain you get from using a pool to manage expensive resource objects outweighs that overhead.

However, be cautious on implementing a resource pool. The following mistakes in pool management are often observed:

- a resource object which should be used only serially is given to more than one user at the same time
- objects that are returned to the pool are not properly accounted for and are therefore not reused, wasting resources and causing a memory leak
- elements or object references kept in the pool are not reset or cleaned up properly before being given to the next user

These mistakes can have severe consequences including data corruption, memory leaks, a race condition, or even a security problem. Our advice in managing your pool is: keep your algorithm simple.

The J2EE section in this document includes examples showing how you can use Oracle's built-in JDBC connection caching and the servlet's `SingleThreadModel` to help manage a shared pool without implementing it yourself.

2.11.2 Recycle Objects

Recycling objects is similar to creating an object pool. But there is no need to manage it because the pool only has one object. This approach is most useful for relatively large container objects (such as `Vector` or `Hashtable`) that you want to use for holding some temporary data. Reusing these objects instead of creating new ones each time can avoid memory allocation and reduce garbage collection.

Similar to using a pool, you must take precautions to clear all the elements in any recycled object before you reuse it to avoid memory leak. The collection interfaces have the built-in `clear()` method that you can use. If you are building your object, you should remember to include a `reset()` or `clear()` method if necessary.

2.11.3 Use Lazy Initialization to Defer Creating the Object Until You Need It.

Defer creating an object until it is needed if the initialization of the object is expensive or if the object is needed only under some specific condition.

```
public class myClass
{
    private mySpecialObject myObj;
    …
    public mySpecialObject
             getSpecialObject()
    {
         if (myObj == null)
             myObj = new mySpecialObject();
         return myObj;
    }
    …
}
```

2.12 Use Stringbuffer Instead of String Concatenation

The `String` class is the most commonly used class in Java. Especially in Web applications, it is used extensively to generate and format HTML content.

`String` is designed to be immutable; in order to modify a `String`, you have to create a new `String` object. Therefore, string concatenation can result in creating many intermediate `String` objects before the final `String` can be constructed. `StringBuffer` is the mutable companion class of `String`; it allows you to modify the `String`. Therefore, `StringBuffer` is generally more efficient than `String` when concatenation is needed.

This section also features the following practices:

- Use StringBuffer Instead of String Concatenation If You Repeatedly Append to a String In Multiple Statements
- Use Either String or StringBuffer If the Concatenation Is Within One Statement
- Use StringBuffer Instead of String Concatenation If You Know the Size of the String

2.12.1 Use StringBuffer Instead of String Concatenation If You Repeatedly Append to a String In Multiple Statements

Using the “+=” operation on a `String` repeatedly is expensive.

For example:

```
String s = new
String();
     [do some work …]
s += s1;
     [do some more work…]
s += s2;
```

Replace the above string concatenation with a StringBuffer:

```
StringBuffer strbuf = new StringBuffer();
     [do some work …]
strbuf.append(s1);
     [so some more work …]
strbuf.append(s2);
String s = strbuf.toString();
```

2.12.2 Use Either String or StringBuffer If the Concatenation Is Within One Statement

`String` and `StringBuffer` perform the same in some cases; so you do not need to use `StringBuffer` directly.

```
    String s = “a” + “b” + “c”;
to
    String s = “abc”;
```

Optimization is done automatically by the compiler.

- The Java2 compiler will automatically collapse the above.
- The Java2 compiler will also automatically convert the following:

```
     String s = s1 + s2;
to
     String s = (new StringBuffer()).append(s1).append(s2).toString();
```

In these cases, there is no need to use `StringBuffer` directly.

2.12.3 Use StringBuffer Instead of String Concatenation If You Know the Size of the String

The default character buffer for `StringBuffer` is 16. When the buffer is full, a new one has to be re-allocated (usually at twice the size of the original one). The old buffer will be released after the content is copied to the new one. This constant reallocation can be avoided if the StringBuffer is created with a buffer size that is big enough to hold the `String`.

The following will be more efficient than using a `String` concatenation.

```
    String s = (new StringBuffer(1024)).
append(s1).append(s2). toString();
```

will be faster than

```
    String s = s1 + s2;
```

3 J2EE Best Practices

This chapter describes the J2EE best practices. The topics include:

- JSP Best Practices
- Servlet Best Practices
- Sessions Best Practices
- EJB Best Practices
- Data Access Best Practices
- Java Message Service Best Practices
- Web Services Best Practices

3.1 JSP Best Practices

This section describes JSP best practices. It includes the following topics:

3.1.1 Pre-Translate JSPs Before Deployment

You can use Oracle’s ojspc tool to pre-translate the JSPs and avoid the translation overhead that has to be incurred when the JSPs are executed the first time. You can pre-translate the JSPs on the production system or before you deploy them. Also, pre-translating the JSPs allows you the option to deploy only the translated and compiled class files, if you choose not to expose and compromise the JSP source files.

3.1.2 Separate Presentation Markup From Java

Separating presentation markup such as HTML from Java code is a good practice to get better performance from your application. The following are a few tips:

- Use JavaBeans for the business logic and JSPs only for the view. Thus, JSPs should primarily contain logic for HTML (or other presentation markup) generation only.
- Use stylesheets when appropriate to provide even more separation of the aspects of HTML that a user can control better.
- JSPs containing a large amount of static content, including large amounts of HTML code that does not change at runtime, which may result in slow translation and execution. Use dynamic includes, or better, enable the external resource configuration parameter to put the static HTML into a Java resource file.

3.1.3 Use JSP Template Mechanism

Using the JSP code `out.print("<html>")` requires more resources than including static template text. For performance reasons, it is best to reserve the use of `out.print()` for dynamic text.

3.1.4 Set Sessions=False If Not Using Sessions

The default for JSPs is session=”true”. If your JSPs do not use any sessions, you should set session=”false” to eliminate the overhead of creating and releasing these internal sessions created by the JSP runtime. To disable sessions, set the directive as follows:

```
<%@page session=”false” %>
```

3.1.5 Always Invalidate Sessions When No Longer Used

Sessions add performance overhead to your Web applications. Each session is an instance of the `javax.servlet.http.HttpSession` class. The amount of memory used per session depends on the size of the session objects created.

If you use sessions, ensure that you explicitly cancel each session using the `invalidate()` method to release the memory occupied by each session when you no longer need it.

The default session timeout for OC4J is 30 minutes. You can change this for a specific application by setting the `<session-timeout>` parameter in the `<session-config>` element of `web.xml`.

3.1.6 Set Main_Mode Attribute To "justrun"

This attribute, found in `global-web-application.xml`, determines whether classes are automatically reloaded or JSPs are automatically recompiled. In a deployment environment set `main_mode` to `justrun`. The runtime dispatcher does not perform any timestamp checking, so there is no recompilation of JSPs or reloading of Java classes. This mode is the most efficient mode for a deployment environment where code is not expected to change.

If comparing timestamps is unnecessary, as is the case in a production deployment environment where source code does not change, you can avoid all timestamp comparisons and any possible retranslations and reloads by setting the `main_mode` parameter to the value `justrun`. Using this value can improve the performance of JSP applications.

Note that before you set `main_mode` to `justrun`, make sure that the JSP is compiled at least once. You can compile the JSP by invoking it through a browser or by running your application (using the `recompile` value for `main_mode`). This assures that the JSP is compiled before you set the `justrun` flag.

3.1.7 Use Available JSP Tags In Tag Library

JSP tags make the JSP code cleaner, and more importantly, provide easy reuse. In some cases, there is also a performance benefit. Oracle9*i*AS ships with a very comprehensive JSP tag library that will meet most needs. In cases where custom logic is required or if the provided library is insufficient, you can build a custom tag library, if appropriate.

3.1.8 Minimize Context Switching Between Servlets and EJBs

Minimize context switching between different Enterprise JavaBeans (EJB) and servlet components especially when the EJB and Web container processes are different. If context switching is required, co-locate EJBs whenever possible.

3.1.9 Package JSP Files In EAR File For Deployment Rather Than Standalone

Oracle9*i*AS Release 2 supports deploying of JSP files by copying them to the appropriate location. This is very useful when developing and testing the pages. However, this is not recommended for releasing your JSP-based application for production. You should always package JSP files into an Enterprise Archive (EAR) file so that they can be deployed in a standard manner - even across multiple application servers.

3.1.10 Use Compile-Time Object Introspection

Developers should try to rely on compile-time object introspection on the beans and objects generated by the tag library instead of request-time introspection.

3.1.11 Choose Static Versus Dynamic Includes Appropriately

JSP pages have two different include mechanisms:

1. Static includes which have a page directive such as:

   ```
   <%@ include file=”filename.jsp” %>
   ```

2. Dynamic includes which have a page directive such as:

   ```
   <jsp:include page="filename.jsp" flush="true" />
   ```

Static includes create a copy of the include file in the JSP. Therefore, it increases the page size of the JSP, but it avoids additional trips to the request dispatcher. Dynamic includes are analogous to function calls. Therefore, they do not increase the page size of the calling JSP, but they do increase the processing overhead because each call must go through the request dispatcher.

Dynamic includes are useful if you cannot determine which page to include until after the main page has been requested. Note that a page that can be dynamically included must be an independent entity, which can be translated and executed on its own.

3.1.12 Disable JSP Page Buffer If Not Used

In order to allow part of the response body to be produced before the response headers are set, JSPs can store the body in a buffer.

When the buffer is full or at the end of the page, the JSP runtime will send all headers that have been set, followed by any buffered body content. This buffer is also required if the page uses dynamic `contentType` settings, forwards, or error pages. The default size of a JSP page buffer is 8 KB. If you need to increase the buffer size, for example to 20KB, you can use the following JSP attribute and directive:

```
<%@page buffer=”20kb” %>
```

If you are not using any JSP features that require buffering, you can disable it to improve performance; memory will not be used in creating the buffer, and output can go directly to the browser. You can use the following directive to disable buffering:

```
<%@ page buffer="none" %>
```

3.1.13 Use Forwards Instead of Redirects

For JSPs, you can pass control from one page to another by using forward or redirect, but forward is always faster. When you use forward, the forwarded target page is invoked internally by the JSP runtime, which continues to process the request. The browser is totally unaware that such an action has taken place.

When you use redirect, the browser actually has to make a new request to the redirected page. The URL shown in the browser is changed to the URL of the redirected page, but it stays the same in a forward operation.

Therefore, redirect is always slower than the forward operation. In addition, all request scope objects are unavailable to the redirected page because redirect involves a new request. Use redirect only if you want the URL to reflect the actual page that is being executed in case the user wants to reload the page.

3.1.14 Use JSP Tagged Cache

Using the Java Object Cache in JSP pages, as opposed to servlets, is particularly convenient because JSP code generation can save much of the development effort. OracleJSP provides the following tags for using the Java Object Cache:

- `ojsp:cache`
- `ojsp:cacheXMLObj`
- `ojsp:useCacheObj`
- `ojsp:invalidateCache`

Use the `ojsp:cacheXMLObj` or `ojsp:cache` tag to enable caching and specify cache settings. Use `ojsp:useCacheObj` to cache any Java serializable object. Use the `ojsp:invalidateCache` tag to invalidate a cache block. Alternatively, you can arrange invalidation through the `invalidateCache` attribute of the `ojsp:cacheXMLObj` or `ojsp:cache` tag.

3.1.15 Use well_known_taglib_loc To Share Tag Libraries

As an extension of standard JSP "well-known URI" functionality described in the JSP 1.2 specification, the OC4J JSP container supports the use of a shared tag library directory where you can place tag library JAR files to be shared across multiple Web applications. The benefits are:

- avoidance of duplication of tag libraries between applications
- allow easy maintenance as the TLDs can be in a single JAR file
- application size is minimized

OC4J JSP `well_known_taglib_loc` configuration parameter specifies the location of the shared tag library directory. The default location is `j2ee/home/jsp/lib/taglib/` under the *`ORACLE_HOME`* directory. If *`ORACLE_HOME`* is not defined, it is the current directory (from which the OC4J process was started).

The shared directory must be added to the server-wide `CLASSPATH` by specifying it as a library path element. The default location is set in the `application.xml` file in the OC4J configuration files directory (`j2ee/home/config` by default) and can be altered.

3.1.16 Use JSP-Timeout for Efficient Memory Utilization

Resource utilization is a key factor for any efficient application. Oracle9*i*AS 9.0.3 introduces the `<orion-web-app>` attribute `jsp-timeout` that can be specified in the OC4J `global-web-application.xml` file or `orion-web.xml` file. The `jsp-timeout` attribute specifies an integer value, in seconds, after which any JSP page will be removed from memory if it has not been requested. This frees up resources in situations where some pages are called infrequently. The default value is 0, for no timeout.

Like other attributes use the `<orion-web-app>` element of the OC4J `global-web-application.xml` file to apply to all applications in an OC4J instance. To set configuration values to a specific application, use the `<orion-web-app>` element of the deployment-specific `orion-web.xml` file.

3.1.17 Workarounds for the 64K Size Limit for the Generated Java Method

The Java Virtual Machine (JVM) limits the amount of code to 65536 bytes per Java method. Sometimes, as the JSPs grow larger, there is a possibility of hitting this limit. The following are some suggestions to workaround this limitation:

- As a general rule, design smaller JSPs for your web application.
- If your JSP uses tag libraries heavily, and if you are hitting the 64k limit, use the reduce_tag_code config parameter to reduce the size of generated code for custom tag usage. Note that this may impact performance.

3.2 Servlet Best Practices

This section describes servlet best practices. It includes the following topics:

3.2.1 Perform Costly One-Time Operation in Servlet init() Method

Use a servlet's `init()` method to perform any costly one-time initialization operations. Examples include:

1. Setting up resource pools.
2. Retrieving common data from a database that can be cached in the mid-tier to reduce warm-up time.

The `destroy()` method can be used to execute operations that release resources acquired in the `init()` method.

3.2.2 Improve Performance by Loading Servlet Classes at OC4J Startup

By default, OC4J loads a servlet when the first request for it is made. OC4J also allows you to load servlet classes when the JVM that runs the servlet is started. To do this, add the `<load-on-startup>` sub-element to the `<servlet>` element in the application's `web.xml` configuration file.

For example, add the `<load-on-startup>` as follows:

```
<servlet>
<servlet-name>viewsrc</servlet-name>
<servlet-class>ViewSrc</servlet-class>
<load-on-startup>
</servlet>
```

Using the load-on-startup facility increases the start-up time for your OC4J process but decreases first-request latency for servlets.

Using Oracle Enterprise Manager, you can also specify that OC4J load an entire Web module on startup. To specify that a Web module is to be loaded on startup, select the Web site Properties page for an OC4J instance, and then select the Load on Startup checkbox.

3.2.3 Analyze Servlet Duration for Performance Problems

It is useful to know the average duration for servicing servlet and JSP requests in your J2EE enterprise application. By understanding how long a servlet takes to service requests when the system is not under load, you can more easily determine the cause of a performance problem when the system is loaded. The average response time of a given servlet is reported in the metric `service.avg` 1 for that servlet. You should only examine this value after making many calls to the servlet so that any startup overhead such as class loading and database connection establishment is amortized.

As an example, suppose you have a servlet for which you notice the `service.avg` to be 32 milliseconds. And, suppose you notice a response time increase when your system is loaded but not CPU bound. When you examine the value of `service.avg`, you might find that the value is close to 32 ms, in which case you can assume the degradation is probably due to your system or application server configuration rather than your application. If, on the other hand, you notice that `service.avg` has increased significantly, you should look for the problem in your application. For example, multiple users of the application may be contending for the same resources, including but not limited to database connections.

See Also: *Oracle9i Application Server Performance Guide*

3.2.4 Understand Server Request Load When Debugging

In debugging servlet and JSP problems, it is often useful to know how many requests your OC4J processes are servicing. If the problems are performance related, it is always helpful to know if they are aggravated by a high request load. You can track the requests for a particular OC4J instance using Oracle Enterprise Manager or by viewing an application’s Web module metrics.

3.2.5 Find Large Servlets That Require a long Road Time When Debugging

You may find that a servlet application is especially slow the first time it is used after the server is started or that it is intermittently slow. It is possible that when this happens, the server is heavily loaded, and response times are suffering as a result. If there is no indication of a high load, which you can detect by monitoring your access logs, periodically monitoring CPU utilization, or by tracking the number of users that have active requests to the HTTP server(s) and OC4J instance(s), then you may have a large servlet that takes a long time to load.

You can see if you have a slow loading servlet by looking at `service.maxTime`, `service.minTime`, and `service.avg`. If the time to load the servlet is much longer than the time it takes to service the first request after loading, the first user that accesses the servlet after your system is started will feel the delay, and `service.maxTime` will be large. You can avoid this by configuring the system to initialize your servlet when it starts.

3.2.6 Watch for Unused Sessions When Debugging

You should regularly monitor your applications to look for unused sessions. It is easy to inadvertently write servlets that do not invalidate their sessions. Without access to application source code, you may not be aware that it could be causing problems for your production host(s), but sooner or later you may notice higher memory consumption than expected. You can check for unused sessions or sessions which are not being properly invalidated using the session metrics: `sessionActivation.time`, `sessionActivation.completed`, and `sessionActivation.active`.

3.2.7 Watch for Abnormal Session Usage When Debugging

The following is an example that shows an application that creates sessions but never uses them.

The following are metrics for a JSP under `/oc4j/<application>/WEBs/<context>`:

```
session.Activation.active: 500 ops
session.Activation.completed: 0 ops
```

This application created 500 sessions that are all still active. Possibly, this indicates that the application makes unnecessary use of the sessions. Over time, it will cause memory or CPU consumption problems.

A well-tuned application shows `sessionActivation.active` with a value that is less than sessionActivation.completed before the session time out. This indicates that the sessions are probably being used and cleaned up.

Suppose you have a servlet that uses sessions effectively and invalidates them appropriately. Then, you might see a set of metrics such as the following:

```
session.Activation.active: 2 ops
session.Activation.completed: 500 ops
```

The fact that two sessions are active when more than 500 have been created and completed indicates that sessions are being invalidated after use.

3.2.8 Load Servlet Session Security Routines at Startup

OC4J uses the class `java.security.SecureRandom` for secure seed generation. The very first call to this method is time consuming. Depending on how your system is configured for security, this method may not be called until the very first request for a session-based servlet is received. One alternative is to configure the application to load on startup in the application's `web.xml` configuration file and to create an instance of `SecureRandom` during the class initialization of the application. The result will be a longer OC4J startup time in lieu of a delay in servicing the first request.

3.2.9 Retry Failed Transactions and Idempotent HttpServlet.doGet() Exactly Once

Retries are discouraged in general because if every `catch` block of an N frame `try..catch` stack performs M retries, the innermost method gets retried (MN)/2 times. This is likely to be perceived by the end user as a hang, and hangs are worse than receiving an error message.

If you could pick just one `try..catch` block to retry, it would be best to pick the outermost block. It covers the most code, and therefore, also covers the most exceptions. Of course, only idempotent operations should be retried. Transactions guarantee that database operations can be retried as long as the failed try results in a rollback and all finally blocks restore variables to a state consistent with the rolled back database state. Often, the case will be that a servlet's `doGet()` method will perform the retry, and a servlet's `doPost()` method will rollback any existing transaction and retry with a new transaction.

Other cases where a retry is warranted are:

- The semantics of a checked exception suggest a retry using a different method or different parameters. For example, `ShoppingCart.insert()` might throw an `ItemExists` exception, and this should be caught and `ShoppingCart.incrementQuantity()` should be tried.
- Several object replicas exist (usually in different processes). A failure of one (for example, a remote exception) could be caught and another replica could be tried. Retry only if the operation is idempotent. Most catch-all exceptions do not guarantee that all effects from the failed invocation are undone.

For example, if the database tier uses Oracle Real Application Clusters (ORAC), then a new connection may be to any available database server machine that mounts the desired database. For JDBC, the `DataSource.getConnection()` method is usually configured to pick among ORAC machines.

Also review the following:

- Do One-time Resource Allocation and Cleanup in init() and destroy() Methods.

3.2.9.1 Do One-time Resource Allocation and Cleanup in init() and destroy() Methods.

`init()` and `destroy()` methods are only called during the servlet initialization and destruction respectively.

3.2.10 Use HTTP Servlet.doPost() for Requests That Update Database

The HTTP specification states that the GET method should be idempotent and free of side effects. Proxies and caches along the route from client to mid-tier, as well as a user pressing the reload button, could cause the GET method at the mid-tier to be called more than once.

HTTP POST is not assumed to be idempotent. Browsers typically require client confirmation before another POST operation, and intermediate proxies/caches do not retry or cache the result of a POST. However, a failure may require the client to manually retry (press RELOAD or press BACK on the browser and then re-submit), which is not safe unless the update is idempotent.

Hence, it is important to use POST instead of GET for these kinds of updates. Some practices to be aware of are:

- Applications can warn users about potential duplicate requests. This can be implemented by encoding a unique request-id in a hidden form field and writing the request-id of each update request to the database. An update request first compares its request-id with those of already-processed requests in the database and warns the user about a potential duplicate if there is a match. Because the user may have intended to submit two separate and unique updates, the system cannot make duplicate suppression transparent.

Another good practice is to label non-idempotent submit buttons with advice against reloading or re-submitting the current page and provide instructions on which application level logs should be consulted should a failure occur. Because this is easier to do than implementing request-ids, this is a more common practice.

3.2.11 Avoid Duplicating Libraries

Avoid duplicating copies of the same library at different location in your application server. Duplication of class libraries can lead to several classloading problems and may consume additional memory and disk space. If your class library is used by multiple applications, then you can put it at the application server level by using the `<library>` tag in `application.xml`. Or, use the `<parent>` attribute in `server.xml` to share libraries in two applications.

If you have a library that is shared between multiple modules in the same application, i.e. two web modules in the same EAR file, then use the WAR file manifest's CLASSPATH to share the class libraries between the modules instead of duplicating the libraries in the `WEB-INF/lib` for every module. In order to enable the CLASSPATH in a WAR file manifest, the following has to be defined in `orion-web.xml`:

```
<web-app-class-loader include-war-manifest-class-path="true" />
```

3.2.12 Use Resource Loading Appropriately

If you are using dynamic classloading or are loading a resource, for example, properties file in your application, use the correct loader.

If you call `Class.forName()`, always explicitly pass the loader returned by `Thread.currentThread().getContextClassLoader`.

If you are loading a properties file, use `Thread.currentThread().getContextClassLoader().getResourceAsStream()`.

3.3 Sessions Best Practices

This section describes session best practices. It includes the following topics:

- Persist Session State if Appropriate
- Replicate Sessions if Persisting is Not an Option
- Do Not Store Shared Resources in Sessions
- Set Session Timeout Appropriately
- Monitor Session Memory Usage
- Always Use Islands, But Keep Island Size Small
- Use a Mix of Cookie and Sessions
- Use Coarse Objects Inside HTTP Sessions
- Use Transient Data in Sessions Whenever Appropriate
- Invalidate Sessions
- Miscellaneous Guidelines

3.3.1 Persist Session State if Appropriate

HTTP Sessions are used to preserve the conversation state with a browser. As such, they hold information, which if lost, could result in a client having to start over the conversation.

Hence, it is always safe to save the session state in database. However, this imposes a performance penalty. If this overhead is acceptable, then persisting sessions is indeed the best approach.

There are trade-offs when implementing state safety that affect performance, scalability, and availability. If you do not implement state-safe applications, then:

- A single JVM process failure will result in many user session failures. For example, work done shopping online, filling in a multiple page form, or editing a shared document will be lost, and the user will have to start over.
- Not having to load and store session data from a database will reduce CPU overhead, thus increasing performance.
- Having session data clogging the JVM heap when the user is inactive reduces the number of concurrent sessions a JVM can support, and thus decreases scalability. In contrast, a state safe application can be written so that session state exists in the JVM heap for active requests only, which is typically 100 times fewer than the number of active sessions.

To improve performance of state safe applications:

- Minimize session state. For example, a security role might map to detailed permissions on thousands of objects. Rather than store all security permissions as session state, just store the role id. Maintain a cache, shared across many sessions, mapping role id to individual permissions.
- Identify key session variables that change often, and store these attributes in a cookie to avoid database updates on most requests.
- Identify key session variables that are read often, and use `HttpSession` as a cache for that session data in order to avoid having to read it from the database on every request. You must manually synchronize the cache, which requires care to handle planned and unplanned transaction rollback.

3.3.2 Replicate Sessions if Persisting is Not an Option

For the category of applications where the HTTP session state information cannot be persisted and retrieved on each HTTP request (due to the performance overhead), OC4J provides an intermediate option – replication.

It can replicate the session state information across an island of servers (which are in the same cluster). This provides a performance improvement because the sessions remain in memory, and fault tolerance – because Oracle HTTP Server automatically routes the HTTP requests to a different server in the island, if the original OC4J (and the session it contains) is down.

Hence, the best practice here is to at least setup two servers in an island, so that they can back session state for each other.

3.3.3 Do Not Store Shared Resources in Sessions

Objects that are stored in the session objects will not be released until the session times out (or is invalidated). If you hold any shared resources that have to be explicitly released to the pool before they can be reused (such as a JDBC connection), then these resources may never be returned to the pool properly and can never be reused.

3.3.4 Set Session Timeout Appropriately

Set session timeout appropriately (`setMaxInactiveInterval()`) so that neither sessions timeout frequently nor does it live for ever this consuming memory.

3.3.5 Monitor Session Memory Usage

Monitor the memory usage for the data you want to store in session objects. Make sure there is sufficient memory for the number of sessions created before the sessions time out.

3.3.6 Always Use Islands, But Keep Island Size Small

Setting up an island of OC4J JVMs causes the sessions to be replicated across all JVMs. This provides better fault tolerance, since a server crash does not necessarily result in a lost session. Oracle9*i*AS automatically re-routes request to another server in the island – thus an end-user never finds out about a failure.

However, this replication overhead increases as more servers are added to the island. For example: if your session object requires 100KB per user, and there are 100 users per server. This results in a 10MB memory requirement for session replication per server. If you have 5 servers in an island, the memory requirement jumps five-fold. Since islands provide session replication, it is, in general, not prudent to exceed an island size beyond 3.

Hence, setting up multiple islands, with few servers in an island is a better choice compared to having a fewer number of larger sized islands.

3.3.7 Use a Mix of Cookie and Sessions

Typically, a cookie is set on the browser (automatically by the container), to track a user session. In some cases, this cookie may last a much longer duration than a single user session. (Example: one time settings, such as to determine the end-user's geographic location).

Thus, a cookie that persists on the client's disk could be used to save information valid for the long-term, while a server side session will typically include information valid for the short-term.

In this situation, the long-term cookie should be parsed on only the first request to the server – when a new session established. The session object created on the server should contain all the relevant information, so as not to require re-parsing the cookie on each request.

A new client side cookie should then be set that contains only an id to identify the server side session object. This is automatically done for any JSP page that uses sessions.

This gives performance benefit since the session object contents do not have to be re-created from the long-term cookie. The other option is of course to save the user settings in a database on the server, and have the user login. The unique userid can then be used to retrieve the contents from the database and store the information in a session.

3.3.8 Use Coarse Objects Inside HTTP Sessions

Oracle9*i*AS automatically replicates sessions when session object is updated. If a session object contains granular objects, for example a person's name), it results in too many update events to all the servers in the island.

Hence, it is recommended to use coarse objects, (for example the person object, as opposed to the name attribute), inside the session.

3.3.9 Use Transient Data in Sessions Whenever Appropriate

Oracle9*i*AS does not replicate transient data in a session across servers in the island. This reduces the replication overhead (and also the memory requirements). Hence, use transient type liberally.

3.3.10 Invalidate Sessions

The number of active users is generally quite small compared to the number of users on the system (ex. of the 100 users on a Web site, only 10 may actually be doing something).

A session is typically established for each user on the system, which costs memory.

Simple things – like a logout button - provide opportunity for quick session invalidation and removal. This avoids memory usage growth since the sessions on the system will be closer to the number of active users, as opposed to all those that have not timed out yet.

3.3.11 Miscellaneous Guidelines

- Use sessions as light-weight mechanism by verifying session creation state.
- Use cookies for long-standing sessions.
- Put recoverable data into sessions so that they can be recovered if the session is lost.
- Store non-recoverable data persistently (in file system or in database using JDBC). However, storing every data persistently is an expensive thing. Instead, one can save data in sessions and use `HttpSessionBindingListener` or other events to flush data into persistent storage during session close.
- Sticky vs Distributable Sessions
  - Distributable session data must be serializable, useful for failover, but are expensive, as the data has to be serialized & replicated among peer processes.
  - Sticky sessions affect load-balancing across multiple JVMs, but are less expensive as there is no state replication.

3.4 EJB Best Practices

This section describes EJB best practices. It includes the following topics:

3.4.1 Local vs. Remote vs. Message Driven EJB

EJBs can be local or remote. If you envision calls to an EJB to originate from the same container as the one running the EJB, local EJBs are better since they do not entail the marshalling, unmarshalling, and network communication overhead. The local beans also allow you to pass an object-by-reference, thus, improving performance further.

Remote EJBs allow clients to be on different machines and/or different application server instances to talk to them. In this case, it is important to use the value object pattern to improve performance by reducing network traffic.

If you choose to write an EJB, write a local EJB over a remote `EJBObject`. Since the only difference is in the exception on the `EJBObject`, almost all of the implementation bean code remains unchanged.

Additionally, if you do not have a need for making synchronous calls, message driven beans are more appropriate.

3.4.2 Decide EJB Use Judiciously

An EJB is a reusable component backed by component architecture with several useful services: persistence, transactions security, naming, etc. However, these additions make it "heavy."

If you just require abstraction of some functionality and are not leveraging the EJB container services, you should consider using a simple JavaBean, or implement the required functionality using JSPs or servlets.

3.4.3 Use Service Locator Pattern

Most J2EE services and/or resources require “acquiring” a handle to them via an initial Java Naming and Directory Interface (JNDI) call. These resources could be an `EJBHomeObject`, or, a JMS topic.

This results in expensive calls to the server machine to resolve the JNDI reference, even though the same client may have gone to the JNDI service for a different thread of execution to fetch the same data!

Hence, it is recommended to have a “Service Locator”, which in some sense is a local proxy for the JNDI service, so that the client programs talk to the local service locator, which in turn talks to the real JNDI service, and that only if required.

The Java Object Cache bundled with the product may be used to implement this pattern.

This practice improves availability since the service locator can hide failures of the backend server or JNDI tree by having cached the lookup. Although this is only temporary since the results still have to be fetched.

Performance is also improved since trips to the back-end application server are reduced.

3.4.4 Cluster Your EJBs

OC4J in Oracle9*i*AS Release 2 provides a mechanism to cluster EJBs. Leveraging this mechanism gives significant benefits:

1. **Load Balancing**: The EJB client(s) are load balanced across the servers in the EJB cluster.
2. **Fault Tolerance**: The state (in case of stateful session beans) is replicated across the OC4J processes in the EJB cluster. If the proxy classes on the client cannot connect to an EJB server, they will attempt to connect to the next server in the cluster. The client does not see the failure.
3. **Scalability:** Since multiple EJB servers behaving as one can service many more requests than a single EJB server, a clustered EJB system is more scalable. The alternative is to have stand-alone EJB systems, with manual partitioning of clients across those servers. This is difficult to configure and does not have fault tolerance advantages.

3.4.5 Cluster Servlets and EJB into Identical Islands

Both servlets and EJBs in OC4J support session state replication, through islands.

An island is a group of servers configured for state replication. A servlet (or JSP) island could be different from an EJB island. Thus you could have a group of servers within an EJB island (sometimes also referred to as an EJB cluster), and a group of servers within a JSP/servlet island. However, this gets confusing and the benefits are fewer.

Hence, it is recommended to configure deployments so as to have a servlet island identical to an EJB island.

Note that to leverage EJB clustering fully, you will need to use remote EJBs, which have some performance implications over local EJBs (discussed in earlier best practice). If you use local EJBs and save a reference to them in a servlet (or JSP) session, when the session is replicated this reference is not valid. It is important to be aware of this trade-off.

3.4.6 Index Secondary Finder Methods

When finder methods, other than `findByPrimaryKey` and `findAll`, are created they may be extremely inefficient if appropriate indexes are not created that help to optimize execution of the SQL generated by the container.

3.4.7 Understand EJB Lifecycle

As a developer, it is imperative that you understand the EJB lifecycle. Many problems can be avoided by following the lifecycle and the expected actions during call backs more closely.

This is especially true with entity beans and stateful session beans. An example might be: in a small test environment during testing, a bean may never get passivated, and thus a mis-implementation (or non-implementation) of `ejbPassivate()` and `ejbActivate()` may not show up until later. Moreover, since these are not used for stateless beans, they may confuse new developers.

3.4.8 Use Deferred Database Constraints

For those constraints that may be invalid for a short time during a transaction but will be valid at transaction boundaries, use deferred database constraints. For example, if a column is not populated during an `ejbCreate()`, but will be set prior to the completion of the transaction, then you may want to set the not null constraint for that column to be deferred. This also applies to foreign key constraints that are mirrored by EJB relationships with EJB 2.0.

3.4.9 Create a Cache with Read Only EJBs

For those cases where data changes very slowly or not at all, and the changes are not made by your EJB application, read-only beans may make a very good cache. A good example of this is a country EJB. It is unlikely that it will change very often and it is likely that some degree of stale data is acceptable.

To do this:

1. Create read-only entity beans.
2. Set `exclusive-write-access="true"`.
3. Set the validity timeout to the maximum acceptable staleness of the data.

3.4.10 Pick an Appropriate Locking Strategy

It is critical that an appropriate locking strategy be combined with an appropriate database isolation mode for properly performing and highly reliable EJB applications.

Use optimistic locking where the likelihood of conflict in updates is low. If a lost update is acceptable or cannot occur because of application design, use an isolation mode of read-committed. If the lost updates are problematic, use an isolation mode of serializable.

Use pessimistic locking where there is a higher probability of update conflicts. Use an isolation mode of read-committed for maximum performance in this case. Use read-only locking when the data will not be modified by the EJB application.

3.4.11 Understand and Leverage Patterns

With the wider industry adoption, there are several common (and generally) acceptable ways of solving problems with EJBs. These have been widely published in either books or discussion forums, etc. In some sense, these patterns are best practices for a particular problem. These should be researched and followed.

Here are some examples:

- Session Façade: Combines multiple entity bean calls into a single call on a session bean, thus reducing the network traffic.
- Message Façade: Use MDBs if you do not need a return status from your method invocation.
- Value Object Pattern: A value object pattern reduces the network traffic by combining multiple data values that are usually required to be together, into a single value object.

A full discussion on the large number of patterns available is outside the scope of this document, but the references section contains some useful books and / or Web sites on this subject.

3.4.12 When Using Entity Beans, Use Container Managed Aged Persistence Whenever Possible

Although there are some limitations to container-managed persistence (CMP), CMP has a number of benefits. One benefit is portability. With CMP, decisions like persistence mapping and locking model selection become a deployment activity rather than a coding activity. This allows deployment of the same application in multiple containers with no change in code. This is commonly not true for Bean Managed Persistence (BMP) since SQL statements and concurrency control must be written into the entity bean and are therefore specific to the container and/or the data store.

Another benefit is that, in general, J2EE container vendors provide quality of service (QoS) features such as locking model variations, lazy loading, and performance and scalability enhancements, which may be controlled via deployment configuration rather than by writing code. Oracle9*i*AS includes features such as read-only entity beans, minimal writing of changes, and lazy loading of relations, which would have to be built into code for BMP.

A third benefit of CMP is container-managed relationships. Through declarations, not unlike CMP field mapping, a CMP entity bean can have relationships between two entity beans managed by the container with no implementation code required from application developers.

Last but least, tools are available to aid in the creation of CMP entity beans so that minimal work is required from developers for persistence. This allows developers to focus on business logic, which allows them to be more efficient. JDeveloper9i is a perfect example where, through modeling tools and wizards, very little work is required to create CMP entity beans including creation of both the generic EJB descriptors and the Oracle9*i*AS specific descriptors.

Overall, there are cases where CMP does not meet the requirements of an application, but the development effort saved, and the optimizations that J2EE containers like OC4J provide make CMP much more attractive than BMP.

3.5 Data Access Best Practices

This section describes data access best practices. It includes the following topics:

- Datasources Connections Caching and Handling
- Datasource Initialization
- Disable Auto-Commit Mode for Better Performance
- Disable Escape Processing for Better Performance
- Defining Column Types
- Prefetching Rows Improves Performance
- Update Batching Improves Performance
- Use Emulated Data Sources for Better Performance
- Use Emulated and Non-Emulated Data Sources Appropriately
- Use the EJB-Aware Location Specified in Emulated Data Sources
- Set the Maximum Open Connections in Data Sources
- Set the Minimum Open Connections in Data Sources
- Setting the Cache Connection Inactivity Timeout in Data Sources
- Set the Wait for Free Connection Timeout in Data Sources
- Set the Connection Retry Interval in Data Sources
- Set the Maximum Number of Connection Attempts in Data Sources
- Use JDBC Connection Pooling and Connection Caching
- Use JDBC Statement Caching
- Avoid Using More Than One Database Connection Simultaneously in the Same Request
- Tune the Database and SQL Statements

3.5.1 Datasources Connections Caching and Handling

Connections must not be closed within `finalize()` methods. This can cause the connection cache to run out of connections to use, since the connection is not closed until the object that obtained it is garbage collected.

The current connection cache does not provide any mechanism to detect "abandoned" connections, reclaim them, and return them to the cache. All connections must be explicitly closed by the application.

If a connection is declared as static, then it is possible that the same connection object is used on different threads at the same time. Do not declare connections as static objects.

Use the `FIXED_WAIT_SCHEME` when using the connection cache, especially when writing Web applications. This guarantees enforcement of the `MaxLimit` on the connection cache as well as retrieval of a connection from the cache when a connection is returned to the cache.

Always use Connection Cache Timeouts such as `CacheInactivityTimeout` to close unused physical connections in the cache and cause "shrinking" of the cache, thus releasing valuable resources.

Also review the following:

- DataSource Connection Caching Strategies

3.5.1.1 DataSource Connection Caching Strategies

In order to minimize the lock up of resources for long periods of time but allow for recycling of connections from the connection cache, you should use the most appropriate strategy for obtaining and releasing connections as follows:

- Many clients, few connections - Open and close a connection in the same method that needs to use the connection. In order to ensure that connections are returned to the pool, all calls to this method should happen within try-catch, try-finally, or try-catch-finally blocks. This strategy is useful when you have a large number of clients sharing a few connections at the cost of the overhead associated with getting and closing each connection.
- Private client pool - Take advantage of the BMP life cycle. Get a connection within `setEntityContext()` and release the connection in `unsetEntityContext()`. Make connections available to all methods by declaring it a member instance.
- Combined strategy - You may take further advantage of BMP life cycle and implement a strategy which combines the two above.

3.5.2 Datasource Initialization

It is a good practice to put the JNDI lookup of a `DataSource` as part of the application initialization code, since DataSources are simply connection factories.

For example, when using servlets, it is a good idea to put the `DataSource` lookup code into the `init()` method of the servlet.

3.5.3 Disable Auto-Commit Mode for Better Performance

Auto-commit mode indicates to the database whether to issue an automatic commit operation after every SQL operation. Being in auto-commit mode can be expensive in terms of time and processing effort if, for example, you are repeating the same statement with different bind variables.

By default, new connection objects are in auto-commit mode. However, you can disable auto-commit mode with the `setAutoCommit()` method of the connection object (either `java.sql.Conection` or `oracle.jdbc.OracleConnection`).

For better application performance, disable auto-commit mode and use the `commit()` or `rollback()` method of the connection object to manually commit or rollback your transaction.

The following example illustrates how to do this. It assumes you have imported the `oracle.jdbc.*` and `java.sql.*` interfaces and classes.

```
//ds is a DataSource object
Connection conn = ds.getConnection();
// It's faster when auto commit is off
conn.setAutoCommit (false);
// Create a Statement
Statement stmt = conn.createStatement ();
...
```

3.5.4 Disable Escape Processing for Better Performance

Escape processing for SQL92 syntax is enabled by default, which results in the JDBC driver performing escape substitution before sending the SQL code to the database. If you want the driver to use regular Oracle SQL syntax, which is more efficient than SQL92 syntax and escape processing, then disable escape processing using the following statement:

```
stmt.setEscapeProcessing(false);
```

3.5.5 Defining Column Types

Defining column types provides the following benefits:

- Saves a roundtrip to the database server.
- Defines the datatype for every column of the expected result set.
- For VARCHAR, VARCHAR2, CHAR and CHAR2, specifies their maximum length.

The following example illustrates the use of this feature. It assumes you have imported the oracle.jdbc.* and java.sql.* interfaces and classes.

```
//ds is a DataSource object
Connection conn = ds.getConnection();
PreparedStatement pstmt = conn.prepareStatement("select empno, ename, hiredate from emp");

//Avoid a roundtrip to the database and describe the columns
((OraclePreparedStatement)pstmt).defineColumnType(1,Types.INTEGER);

//Column #2 is a VARCHAR, we need to specify its max length
((OraclePreparedStatement)pstmt).defineColumnType(2,Types.VARCHAR,12);
((OraclePreparedStatement)pstmt).defineColumnType(3,Types.DATE);
ResultSet rset = pstmt.executeQuery();
while (rset.next())
System.out.println(rset.getInt(1)+","+rset.getString(2)+","+rset.getDate(3));
pstmt.close();
…
```

3.5.6 Prefetching Rows Improves Performance

Row prefetching improves performance by reducing the number of round trips to a database server. For most database-centric applications, Oracle recommends the use of row prefetching as much as possible. The recommended prefetch size is 10.

The following example illustrates the use of row prefetching. It assumes you have imported the `oracle.jdbc.*` and `java.sql.*` interfaces and classes.

```
//ds is a DataSource object Connection conn = ds.getConnection();

//Set the default row-prefetch setting for this connection
((OracleConnection)conn).setDefaultRowPrefetch(7);

//The following statement gets the default row-prefetch value for

//the connection, that is, 7 Statement stmt = conn.createStatement();

//Subsequent statements look the same, regardless of the row

//prefetch value. Only execution time changes.
ResultSet rset = stmt.executeQuery("SELECT ename FROM emp");
System.out.println( rset.next () );
while( rset.next () )
System.out.println( rset.getString (1) );

//Override the default row-prefetch setting for this

//statement
( (OracleStatement)stmt ).setRowPrefetch (2);
ResultSet rset = stmt.executeQuery("SELECT ename FROM emp");
System.out.println( rset.next () );
while( rset.next() )
System.out.println( rset.getString (1) );
stmt.close();
…
```

3.5.7 Update Batching Improves Performance

Update Batching sends a batch of operations to the database in one trip. When using it:

- Always disable auto-commit mode with Update Batching.
- Use a batch size of around 10.
- Don't mix the standard and Oracle models of Update Batching.

Also review the following:

- Oracle Update Batching
- Standard Update Batching

3.5.7.1 Oracle Update Batching

The following example illustrates how you use the Oracle Update Batching feature. It assumes you have imported the `oracle.driver.*` interfaces.

```
//ds is a DataSource object
Connection conn = ds.getConnection();
//Always disable auto-commit when using update batching
conn.setAutoCommit(false);
PreparedStatement ps =
conn.prepareStatement("insert into dept values (?, ?, ?)");
//Change batch size for this statement to 3
((OraclePreparedStatement)ps).setExecuteBatch (3);
//--------#1------------
ps.setInt(1, 23);
ps.setString(2, "Sales");
ps.setString(3, "USA");
ps.executeUpdate(); //JDBC queues this for later execution
//--------#2------------
ps.setInt(1, 24);
ps.setString(2, "Blue Sky");
ps.setString(3, "Montana");
ps.executeUpdate(); //JDBC queues this for later execution
//--------#3------------
ps.setInt(1, 25);
ps.setString(2, "Applications");
ps.setString(3, "India");
ps.executeUpdate(); //The queue size equals the batch value of
3
//JDBC sends the requests to the database
//--------#1------------
ps.setInt(1, 26);
ps.setString(2, "HR");
ps.setString(3, "Mongolia");
ps.executeUpdate(); //JDBC queues this for later execution
((OraclePreparedStatement)ps).sendBatch(); // JDBC sends the
//queued request
conn.commit();
ps.close();
...
```

3.5.7.2 Standard Update Batching

This example uses the standard “Update Batching” feature. It assumes you have imported the `oracle.driver.*` interfaces.

```
//ds is a DataSource object
Connection conn = ds.getConnection();
//Always disable auto-commit when using update batching
conn.setAutoCommit(false);
Statement s = conn.createStatement();
s.addBatch("insert into dept values ('23','Sales','USA')");
s.addBatch("insert into dept values ('24','Blue
Sky','Montana')");
s.addBatch("insert into dept values
('25','Applications','India')");
//Manually execute the batch
s.executeBatch();
s.addBatch("insert into dept values ('26','HR','Mongolia')");
s.executeBatch();
conn.commit();
ps.close();
...
```

3.5.8 Use Emulated Data Sources for Better Performance

For speed and performance reasons emulated data sources are preferred over non-emulated ones.

A non-emulated datasource provides JDBC v2.0 compliance and additional capabilities such as XA which may not be required for all applications.

3.5.9 Use Emulated and Non-Emulated Data Sources Appropriately

Some of the performance related configuration options have different affects, depending on the type of the data source. OC4J supports two types of data sources, emulated and non-emulated.

The pre-installed default data source is an emulated data source. Emulated data sources are wrappers around Oracle data sources. If you use these data sources, your connections are extremely fast, because they do not provide full XA or JTA global transactional support. Oracle recommends that you use these data sources for local transactions or when your application requires access or update to a single database. You can use emulated data sources for Oracle or non-Oracle databases. You can use the emulated data source to obtain connections to different databases by changing the values of the url and connection-driver parameters.

The following is a definition of an emulated data source:

```
<data-source
class="com.evermind.sql.DriverManagerDataSource"
name="OracleDS"
location="jdbc/OracleCoreDS"
xa-location="jdbc/xa/OracleXADS"
ejb-location="jdbc/OracleDS"
connection-driver="oracle.jdbc.driver.OracleDriver"
username="scott"
password="tiger"
url="jdbc:oracle:thin:@localhost:5521:oracle"
inactivity-timeout="30"
/>
```

Non-emulated data sources are pure Oracle data sources. These are used by applications that want to coordinate access to multiple sessions within the same database or to multiple databases within a global transaction.

3.5.10 Use the EJB-Aware Location Specified in Emulated Data Sources

Each data source is configured with one or more logical names that allow you to identify the data source within J2EE applications. The `ejb-location` is the logical name of an EJB data source. In addition, use the `ejb-location` name to identify data sources for most J2EE applications, where possible, even when not using EJBs. The `ejb-location` only applies to emulated data sources. You can use this option for single phase commit transactions or emulated data sources.

Using the `ejb-location`, the data source manages opening a pool of connections, and manages the pool. Opening a connection to a database is a time-consuming process that can sometimes take longer than the operation of getting the data itself. Connection pooling allows client requests to have faster response times, because the applications do not need to wait for database connections to be created. Instead, the applications can reuse connections that are available in the connection pool.

Oracle recommends that you only use the `ejb-location` JNDI name in emulated data source definitions for retrieving the data source. For non-emulated data sources, you must use the location JNDI name.

3.5.11 Set the Maximum Open Connections in Data Sources

The `max-connections` option specifies the maximum number of open connections for a pooled data source. To improve system performance, the value you specify for the number `max-connections` depends on a combination of factors including the size and configuration of your database server, and the type of SQL operations that your application performs. The default value for `max-connections` and the handling of the maximum depends on the data source type, emulated or non-emulated.

For emulated data sources, there is no default value for `max-connections`, but the database configuration limits that affect the number of connections apply. When the maximum number of connections, as specified with `max-connections`, are all active, new requests must wait for a connection to be become available. The maximum time to wait is specified with wait-timeout.

For non-emulated data sources, there is a property, `cacheScheme`, that determines how max-connections is interpreted. The following lists the values for the `cacheScheme` property (`DYNAMIC_SCHEME` is the default value for `cacheScheme`).

FIXED_WAIT_SCHEME: In this scheme, when the maximum limit is reached, a request for a new connection waits until another client releases a connection.

FIXED_RETURN_NULL_SCHEME: In this scheme, the maximum limit cannot be exceeded. Requests for connections when the maximum has already been reached return null.

For some applications you can improve performance by limiting the number of connections to the database (this causes the system to queue requests in the mid-tier).

For example, for one application that performed a combination of updates and complex parallel queries into the same database table, performance was improved by over 35% by reducing the maximum number of open connections to the database by limiting the value of `max-connections`.

3.5.12 Set the Minimum Open Connections in Data Sources

The `min-connections` option specifies the minimum number of open connections for a pooled data source.

For applications that use a database, performance can improve when the data source manages opening a pool of connections, and manages the pool. This can improve performance because incoming requests don't need to wait for a database connection to be established; they can be given a connection from one of the available connections, and this avoids the cost of closing and then reopening connections.

By default, the value of min-connections is set to 0. When using connection pooling to maintain connections in the pool, specify a value for `min-connections` other than 0.

For emulated and non-emulated data sources, the `min-connections` option is treated differently.

For emulated data sources, when starting up the initial `min-connections` connections, connections are opened as they are needed and once the `min-connections` number of connections is established, this number is maintained.

For non-emulated data sources, after the first access to the data source, OC4J then starts the `min-connections` number of connections and maintains this number of connections.

Limiting the total number of open database connections to a number your database can handle is an important tuning consideration. You should check to make sure that your database is configured to allow at least as large a number of open connections as the total of the values specified for all the data sources `min-connections` options, as specified in all the applications that access the database.

3.5.13 Setting the Cache Connection Inactivity Timeout in Data Sources

The `inactivity-timeout` specifies the time, in seconds, to cache unused connections before closing them.

To improve performance, you can set the `inactivity-timeout` to a value that allows the data source to avoid dropping and then re-acquiring connections while your J2EE application is running.

The default value for the `inactivity-timeout` is 60 seconds, which is typically too low for applications that are frequently accessed, where there may be some inactivity between requests. For most applications, to improve performance, Oracle recommends that you increase the `inactivity-timeout` to 120 seconds.

To determine if the default `inactivity-timeout` is too low, monitor your system. If you see that the number of database connections grows and then shrinks during an idle period, and grows again soon after that, you have two options: you can increase the `inactivity-timeout`, or you can increase the min-connections.

3.5.14 Set the Wait for Free Connection Timeout in Data Sources

The wait-timeout specifies the number of seconds to wait for a free connection if the connection pool does not contain any available connections (that is, the number of connections has reached the limit specified with `max-connections` and they are all currently in use).

If you see connection timeout errors in your application, increasing the `wait-timeout` can prevent the errors. The default wait-timeout is 60 seconds.

If database resources, including memory and CPU are available and the number of open database connections is approaching max-connections, you may have limited `max-connections` too stringently. Try increasing `max-connections` and monitor the impact on performance. If there are not additional machine resources available, increasing `max-connections` is not likely to improve performance.

You have several options in the case of a saturated system:

- Increase the allowable wait-timeout.
- Evaluate the application design for potential performance improvements.
- Increase the system resources available and then adjust these parameters.

3.5.15 Set the Connection Retry Interval in Data Sources

The `connection-retry-interval` specifies the number of seconds to wait before retrying a connection when a connection attempt fails.

If the `connection-retry-interval` is set to a small value, or a large number of connection attempts is specified with `max-connect-attempts` this may degrade performance if there are many retries performed without obtaining a connection.

The default value for the `connection-retry-interval` is 1 second.

3.5.16 Set the Maximum Number of Connection Attempts in Data Sources

The `max-connect-attempts` option specifies the maximum number of times to retry making a connection. This option is useful to control when the network is not stable, or the environment is unstable for any reason that sometimes makes connection attempts fail.

If the `connection-retry-interval` option is set to a small value, or a large number of connection attempts is specified with `max-connect-attempts` this may degrade performance if there are many retries performed without obtaining a connection.

The default value for `max-connect-attempts` is 3.

3.5.17 Use JDBC Connection Pooling and Connection Caching

Constant creation and destruction of resource objects can be very expensive in Java. Oracle suggests using a resources pool to share resources that are expensive to create. The JDBC connections are one of the most common resources used in any Web application that requires database access. They are also very expensive to create. Oracle has observed overhead from hundreds of milliseconds to seconds (depending on the load) in establishing a JDBC connection on a mid-size system with 4 CPUs and 2 GB memory.

In JDBC 2.0, a connection-pooling API allows physical connections to be reused. A pooled connection represents a physical connection, which can be reused by multiple logical connections. When a JDBC client obtains a connection through a pooled connection, it receives a logical connection. When the client closes the logical connection, the pooled connection does not close the physical connection. It simply frees up resources, clears the state, and closes any statement objects associated with the instance before the instance is given to the next client. The physical connection is released only when the pooled connection object is closed directly.

The term pooling is extremely confusing and misleading in this context. It does not mean there is a pool of connections. There is just one physical connection, which can be serially reused. It is still up to the application designer to manage this pooled connection to make sure it is used by only one client at a time.

To address this management challenge, Oracle's extension to JDBC 2.0 also includes connection caching, which helps manage a set of pooled connections. It allows each connection cache instance to be associated with a number of pooled connections, all of which represent physical connection to the same database and schema. You can use one of Oracle's JDBC connection caching schemes (dynamic, fixed with no wait, or fixed wait) to determine how you want to manage the pooled connections, or you can use the connection caching APIs to implement your own caching mechanisms.

3.5.18 Use JDBC Statement Caching

Use JDBC statement caching to cache a JDBC `PreparedStatement` or `OracleCallableStatement` that is used repeatedly in the application to:

- prevent repeated statement parsing and recreation
- reduce the overhead of repeated cursor creation

The performance gain will depend on the complexity of the statement and how often the statement has to be executed. Since each physical connection has its own statement cache, the advantage of using statement caching with a pool of physical connections may vary. That is, if you execute a statement in a first connection from a pool of physical connections, it will be cached with that connection. If you later get a different physical connection and want to execute the same statement, then the cache does you no good.

See Also: *Oracle JDBC Developer's Guide and Reference*

3.5.19 Avoid Using More Than One Database Connection Simultaneously in the Same Request

Using more than one database connection simultaneously in a request can cause a deadlock in the database. This is most common in JSPs. First, a JSP will get a database connection to do some data accessing. But then, before the JSP commits the transaction and releases the connection, it invokes a bean which gets its own connection for its database operations. If these operations are in conflict, they can result in a deadlock.

Furthermore, you cannot easily roll back any related operations if they are done by two separate database connections in case of failure.

Unless your transaction spans multiple requests or requires some complex distributed transaction support, you should try to use just one connection at a time to process the request.

3.5.20 Tune the Database and SQL Statements

Current Web applications are still very database-centric. From 60% to 90% of the execution time on a Web application can be spent in accessing the database. No amount of tuning on the mid-tier can give significant performance improvement if the database machine is saturated or the SQL statements are inefficient.

Monitor frequently executed SQL statements. Consider alternative SQL syntax, use PL/SQL or bind variables, pre-fetch rows, and cache rowsets from the database to improve your SQL statements and database operations. See Oracle's Server Tuning Guide for more information.

Web applications often access a database at the backend. One must carefully optimize handling of database resources, since a large number of concurrent users and high volumes of data may be involved. Database performance tuning can be divided into two categories:

- Tuning of SQL tables and statements
- Tuning of JDBC calls to access the SQL database

Also refer to the following JDBC tuning topics:

- JDBC Tuning
- JDBC Connection Caching
- JDBC Statement Caching
- JDBC Cached Rowsets

3.5.20.1 JDBC Tuning

JDBC objects such Connections, Statements, and Result Sets are quite often used for database access in Web applications. Frequent creation & destruction of these objects can be quite detrimental to the performance and scalability of the application as these objects are quite heavy-weight. So it is always desirable to cache these JDBC resources.

3.5.20.2 JDBC Connection Caching

- Reuse database connections thus avoiding frequent session creations and tear-downs.
- EJBs, servlets, JSPs can use / share the connection cache within a JVM.
- Create at startup as singleton object so that they can be shared across multiple requests.

3.5.20.3 JDBC Statement Caching

- Avoids cursor creation and teardown
- Avoid cursor parsing
- Two types of statement caching:
  - Implicit: Saves Metadata of cursor but clears the State and Data content of the cursor across calls
  - Explicit: Saves Metadata, Data, and State of the cursor across calls
- Can be used with Pooled Connection and Connection Cache
- For example: `conn.setStmtCacheSize`(<cache-size>)

3.5.20.4 JDBC Cached Rowsets

- Result set implementation that is disconnected, serializable, and scrollable
- Free up connections and cursors faster
- Local scrolling on cached data
- Specially useful for:
  - small read-only data set
  - scrolling for long time

3.6 Java Message Service Best Practices

This section describes Java message service (JMS) best practices. It include the following topics:

3.6.1 Set the Correct time_to_live Value

JMS message expiration is set in the `JMSExpiration` header field. If this value is set to zero (the default), then the message will never expire. If the amount of used table space (memory for OC4J) is a concern, then optimally setting the `time_to_live` parameter will keep messages from accumulating. This is especially true in the publish-subscribe domain where messages may sit forever waiting for the final durable subscriber to return to retrieve the message.

3.6.2 Do Not Grant Execute Privilege of the AQ PL/SQL Package to a User or Role While There Are Outstanding OJMS Session Blocking on a Dequeue Operation

This might cause the granting operation to be blocked and even time-out. Granting calls should be executed before other OJMS operations.

Another way to avoid the blocking or time out is to grant roles instead of granting specific privileges to the user directly. AQ has an `AQ_ADMINISTRATOR_ROLE` that can be used, or users may create their own tailored role. You can then grant the execute privilege of a PL/SQL package to this role. Provided the role was created before hand, the granting of the role to the user does not require a lock on the package. This will allow the granting of the role to be executed concurrently with any other OJMS operation.

3.6.3 Close JMS Resources No Longer Needed

When JMS objects like JMS connections, JMS sessions, and JMS consumers are created, they acquire and hold on to server-side database and client-side resources. If JMS programs do not close JMS objects when they are done using them either during the normal course of operation or at shutdown, then database and client-side resources held by JMS objects are not available for other programs to use. The JVM implementation does not guarantee that finalizers will kick in and clean-up JMS object held resources in a timely fashion when the JMS program terminates.

3.6.4 Reuse JMS Resources Whenever Possible

JMS objects like JMS connections are heavy weight and acquire database resources not unlike JDBC connection pools. Instead of creating separate JMS connections based on coding convenience, it is recommended that a given JMS client program create only one JMS connection against a given database instance for a given connect string and reuse this JMS connection by creating multiple JMS sessions against it to perform concurrent JMS operations.

JMS administrable objects like queues, queue tables, durable subscribers are costly to create and lookup. This is because of the database round trips and in some cases, JDBC connection creation and teardown overhead. It is recommended that JMS clients cache JMS administrable objects once they are created or looked up and reuse them rather than create or look them up each time the JMS client wants to enqueue or dequeue a message. The Oracle9*i*AS Java Object Cache could be used to facilitate this caching.

3.6.5 Use Debug Tracing to Track Down Problems

OJMS allows users to turn debug tracing by setting `oracle.jms.traceLevel` to values between 1 and 5 (1 captures fatal errors only and 5 captures all possible trace information including stack traces and method entries and exits). Debug tracing allows one to track down silent or less understood error conditions.

3.6.6 Understand Handle/Interpret JMS Thrown Exceptions

OJMS is required by the JMS specification to throw particular JMS defined exceptions when certain error/exception conditions occur. In some cases the JMS specification allows or expects OJMS to throw runtime exceptions when certain conditions occur. The JMS client program should be coded to handle these conditions gracefully.

The catch all JMS exception, `JMSException`, that OJMS is allowed to throw in certain error/exception cases provides information as to why the error/exception occurred as a linked exception in the `JMSException`. JMS programs should be coded to obtain and interpret the linked exception in some cases.

For instance, when resources like processes, cursors, or tablespaces run out or when database timeouts/deadlocks occur, SQL exceptions are thrown by the backend database, which are presented to the JMS client program as linked SQL exceptions to the catch all `JMSException` that is thrown. It would be useful for JMS programs to log or interpret the ORA error numbers and strings so that the administrator of the database can take corrective action.

The code segment below illustrates a way to print both the `JMSException` and its linked Exception:

try

```
{...}
catch (JMSException jms_ex)
{
    jms_ex.printStackTrace();
    if (jms_ex.getLinkedException() != null)
         jms_ex.getLinkedException().printStackTrace();
}
```

3.6.7 Ensure You Can Connect to the Server Machine and Database From the Client Machine

When debugging JMS connection creation problems or problems with receiving asynchronous messages/notifications make sure that you can:

- Ping the database using `tnsping`
- Connect to the database with its connect string using sqlplus
- Are able to resolve the name or the IP address of the server box from the client (by using a simple program that accesses a socket) and vice versa

3.6.8 Tune Your Database Based on Load

OJMS performance is greatly improved by proper database tuning. OJMS performance is dependent on AQ enqueue/dequeue performance. AQ performance will not scale even if you run the database on a box with better physical resources unless the database is tuned to make use of those physical resources.

3.6.9 Make Sure You Tune the OracleOCIConnectionPool

If a JDBC OCI driver is specified when creating a JMS connection, OJMS creates an `OracleOCIConnectionPool` instance from which to obtain the JDBC OCI connections. Depending on the number of JMS session instances that need to be created against the JMS connection and the number of blocking receives that are expected to be performed at a given time against the given JMS connection, the underlying `OracleOCIConnectionPool` instance can be tuned. This is because the `OracleOCIConnectionPool` instance is not self-tuning and is created with a default maximum number of logical connections that can be created from it. Each blocking receive holds onto a logical JDBC OCI connection, and this connection is not available to share. The JMS developer/application, depending on load, can tune the `OracleOCIConnectionPool` instance on-the-fly by obtaining a handle to the `OracleOCIConnectionPool` instance and using its administrative API's.

3.7 Web Services Best Practices

This section describes Web services best practices. It includes the following topics:

- Create Stateless Web Services Instead of Stateful Web Services Whenever Possible
- UDDI Best Practices

3.7.1 Create Stateless Web Services Instead of Stateful Web Services Whenever Possible

For interactive Web sites, one or a combination of three common techniques is typically used to maintain server-side state between page serves: Cookies, URL injection or embedding state (reference) information in hidden form fields. Unfortunately, none of these techniques work reliably with XML Web services. The Web services core technology stack (Simple Object Access Protocol (SOAP), Web Services Description Language (WSDL)) does not define how to maintain state across multiple client invocations. Vendors who support stateful Web services implement this feature using HTTP cookies and Session ID, but using them comes with its own set of challenges.

First, cookies are an optional feature of HTTP and a client can freely choose either not to support them at all or to return them inconsistently (possibly consulting the user for permission). While this may sometimes be desirable for interactive Web sites because of privacy concerns, XML Web services are entirely different by nature because they are a method for coupling systems and not merely a presentation technique. The fact that support for cookies is entirely optional creates a big problem for Web services.

Another problem is that cookies are transport dependent and are valid only for a distinct peer-to-peer connection. Cookies do not work with routing, they do not work with programmatic redirects to Web service replicas (for wide-area load balancing), and they do not work with any transport other than HTTP.

Also, cookie management becomes a major hassle when Web services call other Web services. The cookies returned by any subordinate Web services must be explicitly stored in the main Web service's state, which is, in turn, referenced by a cookie returned to the client. This is even more difficult to manage when the business code that is called from the main Web service and that calls subordinate Web services itself, isn't aware that it is called from within a Web services environment.

3.7.2 UDDI Best Practices

Since the UDDI specification allows for various types of services to be published (for example, phone, fax, directory services) in addition to Web services, there is no defined relationship between a WSDL document and its representation in the registry. The document "Using WSDL in a UDDI Registry" (http://uddi.org/bestpractices.html) recommends a standard convention for representing WSDL using the UDDI information model. For example, a WSDL document may contain service access details because the UDDI model separates service implementation description from access details; the WSDL document should be modified to remove service access details to produce just an interface WSDL document. Service access information can be stored in a bindingTemplate and the URL for the interface WSDL document can be referenced in the overviewDoc entry in a new wsdlSpec tModel. The bindingTemplate can then reference the new tModel; this allows multiple access points to be registered for the same interface WSDL.

There are also other published UDDI.org technical notes recommended for designing applications and configuring registry taxonomies at:

`http://uddi.org/technotes.html`

The description includes:

- Using WSCL in a UDDI Registry 1.02
- Versioning Taxonomy and Identifier Systems
- Providing a Taxonomy for Use in UDDI Version 2

Review the following UDDI best practices:

- Invocation Patterns
- Taxonomy Development

3.7.2.1 Invocation Patterns

In addition to design-time querying when developing a client application, the UDDI registry can also be used for dynamic, run-time querying of Web services for invocation. This is useful for an application that needs to reliably deliver service invocation responses, for example, if the service provider has changed access points or decided to refer all requests to an affiliate provider.

This can be accomplished by caching binding information in the client application upon the initial query for the Web service, and then only re-querying if the service invocation fails. The client application needs to cache the bindingTemplate, and retrieve the associated tModel that references the Web service (i.e. `wsdlSpec tModel`); the application will use this information for subsequent service invocations. Upon an invocation failure, the application should then use the `bindingKey` value and `get_bindingTemplate` call to retrieve a current version of the `bindingTemplate`. If this `bindingTemplate` differs from the cached one, replace the cached version with the new one and retry the call; if the call still fails, return an error. If the `bindingTemplate` is identical to the cached version, return an error, as the service provider needs to be contacted to update the access information in the registry.

Another example for run-time UDDI access is finding an optimal access point for a particular Web service, based on geographic-based metadata or uptime for that service. The client could find all access points that implement a particular Web service, and from that list, cull only the ones that are closest in physical proximity, or that have a certain guarantee on uptime. Another example using the same methodology is that a client could also gather all the responses for a particular Web service by cycling through the associated access points.

3.7.2.2 Taxonomy Development

An important consideration in publishing Web services to a UDDI registry is classification; without this, service descriptions cannot be realistically searched for and retrieved, based on some required criteria. The UDDI specification itself does not specify any set taxonomies; it is up to the UDDI registry host to decide which taxonomies to specify. For example, the Universal Business Registry (UBR) includes four major taxonomies by which business entities, services, bindingTemplate, and tModels can be classified; they are the uddi-org types, NAICS, ISO 3166, and UNSPSC taxonomies. Each taxonomy type is defined as a tModel, while the valid categories and their IDs are defined using the specific UDDI registry vendor's available tools.

There are two types of categorizations specified by UDDI: checked and unchecked. It is useful to understand the trade-offs between the two, and which one to implement for a given registry.

A checked categorization indicates that the registry will validate any keyValue associated with that categorization for any publish or inquiry API call. If that value is not part of the categorization, the API message will return an error message saying that the value is invalid. This type of categorization can help reduce the occurrence of garbage data from being published; also, having a checked categorization allows browsers and tools to present these categories to the user. The downsides of checked categorizations are that effort is required to develop the taxonomy, and they can potentially change over time.

An unchecked categorization has no a priori knowledge of valid category values; all inquiry and publishing API calls using this categorization will succeed and not be validated. The advantage of this is that developing and maintaining taxonomies are outside the scope of the registry. Of course, using this type of categorization allows the user to more likely introduce garbage data into the registry, as well as preventing UDDI registry administration tools from presenting a category hierarchy listing to the user.

Ideally, and by most what can be read about proper XML Web service architecture nowadays, XML Web services are implemented in an entirely stateless manner. This common recommendation is based on the fact that today the term "XML Web Services" is largely equivalent to the combination of XML and the web's core protocol HTTP. HTTP is a fully stateless protocol and the surrounding infrastructure works best of the code that's driven by HTTP is implemented in an HTTP-aware manner.

4 Oracle9*i*AS Framework Best Practices

This chapter describes Oracle9*i*AS framework best practices. The topics include:

- Design Frameworks and Patterns
- BC4J Best Practices
- Java Object Cache Best Practices

4.1 Design Frameworks and Patterns

Freeware such as Struts and Webwork can run on top of Oracle9*i*AS. For features that leverage more of Oracle9*i*AS with closer integration, Oracle9*i*AS and JDeveloper include frameworks such as an MVC framework, UIX, and BC4J.

The selection of an architecture solution is a crucial decision with major impact. A de-facto standard architecture that fulfills the above stated objectives is the Model-View-Controller (MVC) architecture.

The central idea of the MVC architecture is to separate business logic, presentation, and program flow. By using the MVC architecture in a J2EE application, core data access functionality is separated from the presentation and control logic that uses this functionality. Such separation allows multiple views to share the same enterprise data model, which makes supporting multiple clients easier to implement, test, and maintain. The key concepts of MVC architecture are:

- The “model” represents enterprise data and the business rules that govern access to and updates of this data.
- The “model” often serves as a software approximation to a real-world process. This is represented by EJBs or Java Beans in the J2EE framework.
- A “view” renders the content of a model. It accesses enterprise data through the model and specifies how that data should be presented. JSP is typically used for the presentation logic.
- A “controller” translates interactions with the view into actions to be performed by the model. The user actions appear as HTTP GET and POST requests and trigger activating business processes or state changes. The controller is usually a servlet or a JSP.

4.2 BC4J Best Practices

This section describes BC4J best practices and includes:

- Code to Interfaces
- Choose the Right Deployment Configuration
- Use Application Module Pooling for Scalability
- Use Connection Pooling to Optimize Your Use of Database Connections
- Perform Global Framework Component Customization Using Custom Framework Subclasses
- Use SQL-Only and Forward-only View Objects When Possible
- Do Not Let Your Application Modules Get Too Large
- Use the Right Failover Mode
- Use View Row Spillover to Lower the Memory Required to Cache a Large Number of Rows
- Implement Query Conditions At Design TIme If Possible
- Use the Right JDBC Fetch Size
- Turn Off Event Listening in View Object Used in Batch Processes
- Choose the Right Style of Bind Parameters

4.2.1 Code to Interfaces

The business components framework is organized around interfaces in the `oracle.jbo` package, such as `oracle.jbo.ApplicationModule`, `oracle.jbo.ViewObject`, and `oracle.jbo.Row`. By coding your client to these interfaces, rather than the classes that implement them, your code can stay tier-independent. If, for example, you choose to deploy your business components as an EJB, you can use the same client code you used for testing when your business components were deployed locally, because the interfaces are always accessible in the client tier. Your code doesn't need to use a wire protocol; the business components framework handles all issues of cross-tier communication for you.

The interfaces in `oracle.jbo` are implemented by classes in the package `oracle.jbo.server`. For example, `oracle.jbo.ApplicationModule` is implemented by `oracle.jbo.server.ApplicationModuleImpl`. You should not directly call methods on these implementation classes, because if you deploy remotely, these classes will not be available on the client tier. Invoking the classes would prevent you from ever deploying your application remotely. Instead, you should call methods on the interfaces.

Similarly, when you create your own application modules and view links, you should not call methods on their implementation classes. Instead, you should export methods to create custom interfaces that contain them. These interfaces extend `oracle.jbo.ApplicationModule` or `oracle.jbo.ViewObject` and are also always accessible in the client tier. When you downcast a business component, always downcast to your interface and never to your implementation class.

4.2.2 Choose the Right Deployment Configuration

Your application will have the best performance and scalability if you deploy your business components to the Web module with your client. Unless you have strong reasons (such as wanting to use distributed transactions or EJB security features), we recommend Web module deployment of business components over EJB deployment.

Note that both Web module deployment and EJB deployment are fully J2EE-compliant, and the BC4J framework makes it easy to switch between them. You can test your application in both modes to see which gives you the best performance.

4.2.3 Use Application Module Pooling for Scalability

A client can use application module instances from a pool, called **application module pooling**. Application module pooling has these advantages:

- reduces the amount of time to obtain server-side resources
- allows a small number of instances to serve a much larger number of requests
- addresses the requirements of Web applications that must handle thousands of incoming requests
- lets you preserve session state and provides failover support

For example, in the case of a Web application, you may have 1,000 users but you know that only 100 will be using a certain application module at one time. Hence, you can use an application module pool. When a client needs an application module instance, it takes a free one from the pool and releases it to the pool after either committing or rolling back the transaction. Because the instance is pre-created, end users are saved the time it takes to instantiate the application module when they want to perform a task. Typically, Web-based JSP clients use pools. If you want to make sure that the application module pool has a maximum of 100 application module instances, you can customize the default application module pool.

If your client needs to keep track of application module state, we recommend using stateful mode. In a stateful JSP application, the client does not reserve the application module instance, making it available to other clients if the number of application modules exceeds the recycle threshold. State is, instead, preserved in one of two ways: the application module pool returns a client's original application module if the application module has not been recycled, and the pool persists the state of recycled application modules in the database to be available to clients that request them later.

When you release an application module at the end of a user's session, be sure to use stateless (rather than stateful or reserved) release mode. This frees up database space and allows the pool to recycle the application module immediately.

4.2.4 Use Connection Pooling to Optimize Your Use of Database Connections

Opening a connection to a database is a time-consuming process that can sometimes take longer than getting the data itself. The advantage of connection pooling is that clients that need to instantiate a new application module can have faster response times because they are saved the time of creating the database connection. With database connection pooling, they can reuse a connection that another application module instance already created.

4.2.5 Perform Global Framework Component Customization Using Custom Framework Subclasses

Particularly in large organizations, you may want specific functionality shared by all components of a particular type--for example, by all view objects. An architect can create a thin layer of classes such as `MyOrgViewObjectImpl` that implement the desired behavior. Individual developers can extend `MyOrgViewObjectImpl` instead of `ViewObjectImpl`, and you can use the "substitutes" feature to extend `MyOrgViewObjectImpl` in legacy code.

4.2.6 Use SQL-Only and Forward-only View Objects When Possible

Basing a view object on an entity object allows you to use the view object to insert, update, and delete data, and helps keep view objects based on the same data synchronized. However, if your view object is only going to be used for read-only queries, and there is no chance that the data being queried in this view object will have pending changes made through another view object in the same application module, you should use a SQL-only view object that has no underlying entities. This will give you improved performance since rows do not need to be added to an entity cache.

If you are scrolling through data in one direction, such as formatting data for a Web page, or for batch operations that proceed linearly, you can use a forward-only view object. Forward-only mode prevents data from entering the view cache. Using forward-only mode can save memory resources and time, because only one view row is in memory at a time. Note that if the view object is based on one or more entity objects, the data does pass to the entity cache in the normal manner, but no rows are added to the view cache.

4.2.7 Do Not Let Your Application Modules Get Too Large

A root application module should correspond to one task--anything that you would include in a single database transaction. Do not put more view objects or view links than you will need for a particular task in a single application module.

In addition, consider deferring the creation of view links by creating them dynamically with create`ViewLink()`. If you include all view links at design time, the business logic tier will automatically execute queries for all detail view objects when your client navigates through a master view object. Deferring view link creation will prevent the business logic tier from executing queries for detail view objects that you do not yet need.

For example, for a form in which detail rows are displayed only on request (rather than automatically), including a view link at design time would force the business logic tier to automatically execute a query that might well be unnecessary. To prevent this, you should create a view link dynamically when the detail rows are requested. By contrast, for a form in which detail rows are displayed as soon as a master is selected, you should use a view link created at design time to avoid the runtime overhead of calling create`ViewLink()`.

4.2.8 Use the Right Failover Mode

By default, the application module pool supports failover, which saves an application module's state to the database as soon as the application module is checked into the pool. If the business logic tier or the database becomes inoperable in mid-transaction (due to a power failure or system crash, for example), the client will be able to instantiate a new application module with the same state as the lost one, and no work will be lost.

However, some applications do not require this high level of reliability. If you're not worried about loss of work due to server problems, you may want to disable failover. When failover is disabled, the application module's state exists only in memory until it is committed to the database (at which point the application module's state is discarded) or recycled (at which point the state is saved so that the client can retrieve it). By not saving the application module state every time the application module is checked in, failover-disabled mode can improve performance.

4.2.9 Use View Row Spillover to Lower the Memory Required to Cache a Large Number of Rows

While the business logic tier is running, it stores view rows in a cache in memory (the Java heap). When the business logic tier needs to store many rows at once, you need to make sure it doesn't run out of memory. To do so, you can specify that when the number of rows reaches a certain size, the rows "overflow" to your database to be stored on disk. This feature is called **view row spillover**. If your application needs to work with a large query result, view row spillover can help the cache operate more efficiently.

4.2.10 Implement Query Conditions At Design TIme If Possible

You should include any portion of your query condition that you know in advance in the `WHERE` clause field in the View Object wizard. Only use `setWhereClause()` for genuinely dynamic query conditions. Even if your query conditions are genuinely dynamic, you may be able to use parameterized queries instead of the `setWhereClause()`. For example, if your view object needs to execute a query with the `WHERE` clause “`EMPLOYEE_ID=<x>`” for various values of `x`, use a parameterized `WHERE` clause such as “`EMPLOYEE_ID=:1`”. This is more efficient than repeatedly calling `setWhereClause()`.

4.2.11 Use the Right JDBC Fetch Size

The default JDBC fetch size is optimized to provide the best trade-off between memory usage and network usage for many applications. However, if network performance is a more serious concern than memory, consider raising the JDBC fetch size.

4.2.12 Turn Off Event Listening in View Object Used in Batch Processes

In non-interactive, batch processes, there is no reason for view objects to listen for entity object events. Use `ViewObject.setListenToEntityEvents(false)` on such view objects to eliminate the performance overhead of event listening.

4.2.13 Choose the Right Style of Bind Parameters

Oracle-style bind parameters (:1,:2, and so on) are perform better than JDBC-style bind parameters.

There are only two reasons to use JDBC-style bind parameters:

- Use JDBC-style bind parameters if you may use a non-Oracle JDBC driver.
- Use JDBC-style bind parameters if you have more than one occurrence of the same parameter in the `WHERE` clause.

4.3 Java Object Cache Best Practices

This section describes Java object cache best practices and includes the following topics:

- Allow Cacheaccess Objects to be Released in Error Conditions
- Understand or Delegate Ownership When Doing Synchronize
- Set Open File Descriptor Count to 1024 or Higher
- Use System Classloader for Object Cached with Java Object Cache
- Group Messages Take Precedence Over Individual Objects in the Cache
- Understand What Cache Objects Survive Process Termination
- Return Cacheaccess Object to the Pool When Not in Use
- Use 1:1 Correlation Between Cached Object and Cacheaccess Object
- Do Not Share Cacheaccess Object

4.3.1 Allow Cacheaccess Objects to be Released in Error Conditions

If a `CacheAccess.waitForResponse` or `CacheAccess.releaseOwnership` method call times out, it must be called again until it returns successfully. Or, one of the following should be eventually called to free up resources:

- `CacheAccess.cancelResponse`
- `CacheAccess.releaseOwnership` with a time out value of “-1”

4.3.2 Understand or Delegate Ownership When Doing Synchronize

When an object or group is defined as `SYNCHRONIZE`, ownership is required to load or replace the object, but it is not required for general access to the object or to invalidate the object.

4.3.3 Set Open File Descriptor Count to 1024 or Higher

On Solaris by default, a process is allowed only 64 open file descriptors. This is insufficient for most disk cache use. 1024 or greater is a more appropriate value.

4.3.4 Use System Classloader for Object Cached with Java Object Cache

In general, objects stored in the cache should be defined using the system class loader (defined in `CLASSPATH` when the JVM is initialized) rather than in user defined class loaders. Specifically, any objects that will be shared between applications or may be saved or spooled to disk need to be defined in the system `CLASSPATH`. Failure to do so may result in `ClassNotFoundExceptions` or `ClassCastExceptions`.

4.3.5 Group Messages Take Precedence Over Individual Objects in the Cache

When `destroy` or `invalidate` is called on a group, if there are conflicts between local and distributed definitions of the group and objects in the group, distributed wins. That is, if the group is distributed, all objects in the group will be invalidated / destroyed across the entire cache system regardless if the individual objects or associated groups are defined as local. If the group is defined as local, then local objects within the group will only be invalidated locally, distributed objects will be invalidated throughout the entire cache system.

4.3.6 Understand What Cache Objects Survive Process Termination

Local objects (objects not defined with the `Attributes.DISTRIBUTE` flag) saved to disk using the `CacheAccess.save` method will not survive the termination of the process. By definition, local cache objects are only visible to the cache instance they were loaded into. If that cache instance goes away for any reason, the objects it manages, including those on disk, are lost. If an object needs to survive process termination, both the object and the cache need to be defined as `DISTRIBUTE`.

The cache environment, region, group, and object definitions, are local to a cache. They are not saved to disk nor propagated to other caches. This environment should be defined during the initialization of the application.

4.3.7 Return Cacheaccess Object to the Pool When Not in Use

A `CacheAccess` object should always be closed when it is no longer used. The `CacheAccess` objects are pooled. and acquire other cache resources on behalf of users. If access objects are not closed when they are no longer used, these resources will not be returned to the pool and will not be cleaned up until they are garbage collected by the JVM. If `CacheAccess` objects are continually allocated and not closed, they can cause a significant loss in available resources and a consequent degradation in performance.

4.3.8 Use 1:1 Correlation Between Cached Object and Cacheaccess Object

A `CacheAccess` object only holds a reference to one cached object at a time. If multiple cached objects are being accessed concurrently, multiple `CacheAccess` objects should be used. For objects stored in memory, the consequences of not doing this are minor since Java will prevent the cached object from being garbage collected even if the cache believes it is not being referenced.

For objects stored on disks, if the cache reference is not maintained, the underlying file could be removed by another user or by time-based invalidation, causing unexpected exceptions. It is best to always keep the cache reference open as long as the cached object is being used. This allows the cache system to better manage its resources.

4.3.9 Do Not Share Cacheaccess Object

The `CacheAccess` object should not be shared between threads. This object represents a "user" to the caching system. It contains the current state of that user's access to the cache, which object is currently being accessed, which objects are currently "owned", etc. Trying to share the `CacheAccess` object is unnecessary and can result in non-deterministic behavior.

5

Oracle9*i*AS Web Cache Best Practices

This chapter describes Oracle9*i*AS Web Cache best practices. The topics include:

5.1 Use Partial Page Caching Where Possible

Many Web pages, such as portal pages, are composed of fragments with unique caching properties. For these pages, full-page caching is not feasible. However, Oracle9*i*AS Web Cache provides a partial page caching feature that enables each Web page to be divided into a template and multiple fragments that can in turn be further divided into templates and lower level fragments. Each fragment or template is stored and managed independently; a full page is assembled from the underlying fragments upon request. Fragments can be shared among different templates, so that common fragments are not duplicated to waste precious cache space. Sharing can also greatly reduce the number of updates required when fragments expire. Depending on the application, updating a fragment can be cheaper than updating a full page. In addition, each template or fragment may have its own unique caching policies such as expiration, validation, and invalidation, so that each fragment in a full Web page can be cached as long as possible, even when some fragments are not cached or are cached for a much shorter period of time.

Oracle9*i*AS Web Cache uses Edge Side Includes (ESI) to achieve flexible partial-page caching. ESI is a simple markup language for partial-page caching. Applications can mark up HTTP responses with two different kinds of tags, `<esi:inline>` and `<esi:include>`, that define the fragment/template structure in the response.

See Also: Appendix A, "Oracle9iAS Web Cache Best Practices Appendix"

5.2 Use ESI Variables for Improved Cache Hit Ratio for Personalized Pages

Personalized information often appears in Web pages, making them unique for each user. For example, many Web pages contain tens or hundreds of hyperlinks embedding application session IDs.

To resolve this, create your ESI pages with variables. Because variables can be resolved to different pieces of request information or response information, the uniqueness of templates and fragments can be significantly reduced when personal information abounds. This in turn results in better cache hit ratios.

See Also: Appendix A, "Oracle9iAS Web Cache Best Practices Appendix"

5.3 Leverage JESI Over Hand-Generating the ESI Tags

In dynamic applications, you can use ESI tags for better caching in two ways: hand generating the tags, or using Edge Side Includes for Java (JESI) tags. The latter is recommended for two reasons:

- Reducing hand coding errors, which may not be obvious and may result in more cache misses.
- Facilitating only a partial execution of the JSP page when a cache miss request comes for a specific fragment.

Oracle Corporation recommends that you use JESI to write JSP-based applications.

See Also: Appendix A, "Oracle9iAS Web Cache Best Practices Appendix"

5.4 Use esi:inline and esi:include Tags Appropriately

If a partial page fragment can be fetched independently, such as with a `URLPortlet`, it should always be referenced with an `<esi:include>` template. This enables Oracle9*i*AS Web Cache to fetch the fragment independently of the rest of the page or fragments. However, if the fragment cannot be generated independently of the surrounding page, then the `<esi:inline>` tag enables Oracle9*i*AS Web Cache to cache the inline fragment and re-use it in different contexts. This also results in a single update or invalidation message as opposed to requiring multiple invalidation/updates.

See Also: Appendix A, "Oracle9iAS Web Cache Best Practices Appendix"

5.5 Use Basic Invalidation for Single Objects, Advanced for Multiple

When you need to invalidate one object in the cache, send a basic rather than an advanced invalidation request to avoid cache traversal. Advanced invalidation requests should be reserved for invalidation of multiple objects.

Further, use substring matching to invalidate all objects that match a certain criteria to speed up invalidation.

See Also: Appendix A, "Oracle9iAS Web Cache Best Practices Appendix"

5.6 Build Programmatic Invalidation Into Application Logic

Oracle9*i*AS Web Cache is designed for caching highly dynamic Web pages. There are several ways to safeguard the correctness of the cached content. For those cached pages with content that changes following unpredictable actions by Web site users, the most effective way to ensure correct content is to build programmatic invalidation into application logic.

The application Web server and database are two areas that may benefit from embedded programmatic invalidation. On the application Web server, you can build invalidation into CGI's, JSPs and Java servlets using the Oracle9*i*AS Web Cache Java invalidation API `jawc.jar`. `jawc.jar` is located in the `$`*`ORACLE_HOME`*`/webcache/jlib` directory on UNIX and *`ORACLE_HOME`*`\webcache\jlib` directory on Windows. For example, page A displays information about a certain bike in stock. This page is cacheable. Page B provides users a way to reserve one bike for purchase. On the mid-tier there is a Java servlet or JSP to service page B. To make this servlet cache-aware, use `jawc.jar` to invalidate page A.

Similarly, you can build invalidation into PL/SQL applications using the Oracle9*i*AS Web Cache PL/SQL invalidation API `wxvutil.sql` and `wxvappl.sql`. This way, developers can embed the invocation of invalidation PL/SQL procedure calls into the PL/SQL Web page. `wxvutil.sql` and `wxvappl.sql` are located in the `$`*`ORACLE_HOME`*`/webcache/examples` directory on UNIX and *`ORACLE_HOME`*`\webcache\examples` directory on Windows.

In order to facilitate the caching of JSPs, developers can use the JESI custom tag library included with OC4J and Oracle JDeveloper9i. One of the tags is `<jesi:invalidate>`, which enables programmatic invalidation when the JSP engine processes a page containing this tag. For more information about JESI, see the *Oracle9iAS Containers for J2EE JSP Tag Libraries and Utilities Reference*.

Alternatively, developers can tie invalidation logic to database updates. In the same bike example, you can use a PL/SQL invalidation procedure call to invalidate pages that rely on inventory data stored in the database. In other words, you can apply a database trigger to the row or table containing information about the bike.

For more information about invalidation, see Chapter 8, "Administering Oracle9*i*AS Web Cache," of the *Oracle9iAS Web Cache Administration and Deployment Guide*.

5.7 Use Surrogate-Control Headers Instead of Caching Rules

There are two ways to specify the caching properties of an HTTP response using Oracle9*i*AS Web Cache:

- Administrators can configure caching rules using the Oracle9*i*AS Web Cache Manager interface.

  and/or

- Application developers can set caching policies via the `Surrogate-Control` response header. If a given property is set in both a response header and the configuration, the value set by `Surrogate-Control` overrides rules specified in the configuration.

Although caching rules support the setting of more properties than the `Surrogate-Control` header, it is generally more manageable to set properties in the `Surrogate-Control` header whenever possible. For example, if you need to set the expiration policy and the multiple-version property for a document, it is preferable to use the `Surrogate-Control` header.

Caching rules can be less manageable when there are many different categories of cacheable and non-cacheable documents. In these circumstances, you need to carefully define rule selectors and rule priorities so that the appropriate rule is used for any document. Since a `Surrogate-Control` response header is only associated with one response and overrides the configuration, the properties set in `Surrogate-Control` will not be mistakenly replaced by other configuration rules or newly created configuration rules.

On the other hand, sometimes the configuration approach is more convenient. If a small number of rules are sufficient to describe all the caching properties of all documents that Oracle9*i*AS Web Cache can receive from an origin server, then editing the configuration using the Oracle9*i*AS Web Cache Manager administration interface may be simpler than generating `Surrogate-Control` headers for many documents.

Often, a combination of the two approaches is best.

5.8 Improve Performance, Scalabillity, and Availability

Oracle9*i*AS Web Cache improves the scalability, performance and availability of e-business Web sites. Using Oracle9*i*AS Web Cache, your applications benefit from higher throughput, shorter response times and lower infrastructure costs.

- Unlike legacy cache servers that only handle static data, Oracle9*i*AS Web Cache combines caching, compression and assembly technologies to accelerate the delivery of both static and dynamically generated Web content.
- As the first application server to implement Edge Side Includes (ESI), Oracle9*i* Application Server boasts the industry's fastest edge server (Oracle9*i*AS Web Cache), with support for partial-page caching, personalization and dynamic content assembly at the network edge.
- Oracle9*i*AS Web Cache includes patent-pending clustering functionality that increases capacity for content storage and ensures scalability and availability for cacheable content, even when a member cache experiences a failure or is taken offline for maintenance.
- Oracle9*i*AS Web Cache also provides back-end origin server load balancing, failover, and surge protection features that ensure consistent application performance and greater overall reliability.

Using Oracle9*i*AS Web Cache and its ESI features, your business application’s performance can improve by several orders of magnitude with very little development effort. The return on investment is also significant, both in terms of developer resources (you no longer need to build your own dynamic caching solution) and hardware cost savings.

5.9 Use Two CPUs and Consider Deploying on Dedicated Hardware

You can deploy Oracle9*i*AS Web Cache on the same node as the application Web server or on a separate node. When making your decision, consider system resources, such as the number of CPUs. Oracle9*i*AS Web Cache is designed to use one or two CPUs. Because Oracle9*i*AS Web Cache is an in-memory cache, it is rarely limited by CPU cycles. Additional CPUs do not increase performance significantly. However, the speed of the processors is critical; use the fastest CPUs you can afford.

If other resources are competing with Oracle9*i*AS Web Cache for CPU usage, then you should take the requirements of those resources into account when determining the number of CPUs needed. You can derive a significant performance benefit from Oracle9*i*AS Web Cache running on the same node as the application Web server, although a separate node for Oracle9*i*AS Web Cache is often optimal.

For a Web site with more than one Oracle9*i*AS Web Cache instance, consider installing each instance on a separate two-CPU node, either as part of a cache cluster or as standalone instances. When Oracle9*i*AS Web Cache instances are on separate nodes, you are less likely to encounter operating system limitations, particularly in network throughput. For example, two caches on two separate two-CPU nodes are less likely to encounter operating system limitations than two caches on one four-CPU node.

5.10 Configure Enough Memory

To avoid swapping objects in and out of the cache, it is crucial to configure enough memory for the cache. Generally, the amount of memory (maximum cache size) for Oracle9*i*AS Web Cache should be set to at least 256 MB.

To determine the maximum amount of memory required, take the following steps:

1. Determine what objects you want to cache, how many are smaller than 4 KB and how many are larger than 4 KB. Determine the average size of the objects that are larger than 4 KB. Determine the expected peak load; the maximum number of objects to be processed concurrently.
2. Calculate the amount of memory needed based on the number and size of the objects. Chapter 6, “Initial Setup and Configurations,” of the *Oracle9iAS Web Cache Administration and Deployment Guide* provides a formula to use in calculating the amount of memory needed to cache your objects.

5.11 Allocate Sufficient Network Bandwidth

When you use Oracle9*i*AS Web Cache, make sure that each node has sufficient network bandwidth to accommodate the throughput load. Otherwise, the network may be saturated even though Oracle9*i*AS Web Cache has additional capacity. For example, if your application generates more than 100 megabits of data per second, 10/100 Megabit Ethernet will likely be saturated.

If the network is saturated, consider using Gigabit Ethernet rather than 10/100 Megabit Ethernet. Gigabit Ethernet provides the most efficient deployment scenario to avoid network collisions, retransmissions, and bandwidth starvations. Additionally, consider using two separate network cards: one for incoming client requests and one for requests from the cache to the application Web server.

If system monitoring tools reveal that the network is under utilized and throughput is less than expected, check whether or not the CPUs are saturated.

5.12 Set a Reasonable Number of Network Connections

It is important to specify a reasonable number for the maximum connection limit for the Oracle9*i*AS Web Cache server. If you set a number that is too high, performance can be affected, resulting in slower response time. If you set a number that is too low, fewer requests will be satisfied. You must strike a balance between response time and the number of requests processed concurrently.

For information about setting the number of network connections, see the *Oracle9iAS Web Cache Administration and Deployment Guide*.

5.13 Cluster Cache Instances for Better Availability, Scalability, and Performance

To increase the availability, scalability, and performance of your Web site, you can configure multiple instances of Oracle9*i*AS Web Cache to run as members of a cache cluster. A cache cluster is a loosely coupled collection of cooperating Oracle9*i*AS Web Cache instances working together to provide a single logical cache.

Cache clusters provide failure detection and failover of caches, increasing the availability of your Web site. If a cache fails, other members of the cache cluster detect the failure and take over ownership of the cached content of the failed cluster member.

By distributing the Web site's content across multiple Oracle9*i*AS Web Caches, more content can be cached and more client connections can be supported, expanding the capacity of your Web site and improving its performance.

For more information about cache clusters, see Chapter 3, "Cache Clustering," of the *Oracle9iAS Web Cache Administration and Deployment Guide*.

5.14 Optimize Response Time By Tuning Origin Server and Oracle9*i*AS Web Cache Settings

If you have not configured the origin server or the cache correctly, response time may be slower than anticipated. If the origin server is responding more slowly than expected or if the origin server is not responding to requests from the cache because it has reached its capacity, check the origin server and the Oracle9*i*AS Web Cache settings.

First, check the following:

- The origin server configuration, particularly the `MaxClients` parameter limit. The `MaxClients` parameter, which limits the number of clients that can simultaneously connect to the origin server, should be large enough to allow connections (requests) from the caches. See the *Oracle9i Application Server Performance Guide* for more information about setting limits for the origin server.
- The origin server capacity as set through the Oracle9*i*AS Web Cache Manager. See Chapter 6, "Initial Setup and Configuration," of the *Oracle9iAS Web Cache Administration and Deployment Guide* for information about setting origin server capacity.

- The `MaxClients` parameter limit on the origin server should be greater than or equal to the origin server capacity as set through the Oracle9*i*AS Web Cache Manager.

Then, if the origin server is still busier than anticipated, it may mean that the cache cannot process the requests and is routing more requests to the origin server. Check the following Oracle9*i*AS Web Cache settings:

- The number of cache connections (Maximum Incoming Connections) in the Resource Limits page (Cache-Specific Configuration > Resource Limits) of the Oracle9*i*AS Web Cache Manager
- The memory size for the cache (Maximum Cache Size) in the Resource Limits page
- The cache cluster capacity. In a cluster, if cluster capacity is too low, a cache may not receive a response for owned content from a peer cache within the specified interval. As a result, the request is sent to the origin server.

For information on specifying these settings, see the *Oracle9iAS Web Cache Administration and Deployment Guide.*

If these resources are set reasonably, check the following:

- Caching rules. Make sure that you are caching the appropriate objects including popular objects.
- Priority rankings of the caching rules. If you have a large number of rules, parsing of rules will take additional time.

If the settings for the origin server and Oracle9*i*AS Web Cache are set correctly, but the response times are still higher than expected, check system resources, especially:

- Network bandwidth. See Section 5.11, "Allocate Sufficient Network Bandwidth"
- CPU usage. See Section 5.9, "Use Two CPUs and Consider Deploying on Dedicated Hardware"

5.15 Combine Invalidation and Expiration Policies

With expiration rules, cached objects are marked as invalid after a certain amount of time in the cache. Expirations are useful if it can be accurately predicated when content will change on an origin server or database. To prevent documents from remaining in the cache indefinitely, Oracle Corporation recommends creating expiration rules for all cached documents.

With invalidation requests, an HTTP `POST` message specifies which objects to mark as invalid. Invalidation requests are intended for less predictable, more frequently changing content. Send invalidation requests when you know objects have been refreshed on the origin server. Invalidation policies can be automated, as described in Section 5.6, "Build Programmatic Invalidation Into Application Logic".

5.16 Use Invalidation Propagation in a Cluster

In a cache cluster, you can send invalidation messages to a particular cache cluster member that acts as the *invalidation coordinator*. The coordinator can also be chosen at random when invalidation messages are sent via the network load balancer deployed in front of the cluster. The coordinator propagates the invalidation messages to the other cache cluster members.

The benefits of invalidation propagation include data consistency across cluster members and ease of use for the administrator.

However, under the following circumstances, you may want to disable invalidation propagation and send the invalidation messages to each individual member of the cluster:

- When the cluster membership is in flux. For example, as you begin deployment, you may have made changes to cluster members but have not yet propagated the configuration changes to all members. In this case, the invalidation messages are not propagated to the members with different configurations.
- If you do not want to invalidate data for all cluster members. For example, because of time zone differences, you want to send invalidation messages to only some of the cluster members at one time.

Note that if you do not invalidate data for all cluster members, the cached data may become inconsistent. In addition, cluster members may serve stale data, not only in response to requests from clients, but also in response to requests from their peers.

For more information about invalidation propagation, see the *Oracle9iAS Web Cache Administration and Deployment Guide*.

5.17 Route All HTTP and HTTPS Traffic Through Oracle9*i*AS Web Cache

In general, you should route all HTTP and HTTPS requests through Oracle9*i*AS Web Cache. Ensure documents, especially HTTPS documents, are sent to authorized users through careful use of caching rules.

Depending on the application, you may or may not want requests for secure pages to go through the cache. If every HTTPS request is a non-cacheable page that is unique for each user session or is too sensitive for caching a copy, then route this traffic directly to the origin server. Because no traffic will be cached in this case, routing traffic to the origin server avoids extra encryption / decryption processing time at the Oracle9*i*AS Web Cache layer.

5.18 Create Custom Apology Pages

By default, Oracle9*i*AS Web Cache ships and is configured to serve the following two apology pages:

- `network_error.html`: This file is served when Oracle9*i*AS Web Cache encounters network problems while connecting, sending, or receiving a response from an origin server for a cache miss request.
- `busy_error.html`: This file is served when origin server capacity has been reached.

For a production environment, Oracle Corporation advises that you modify the defaults or create entirely new apology pages to be consistent with other error pages generated by your application.

To create or modify default apology page:

1. Create or modify an apology page in `$`*`ORACLE_HOME`*`/webcache/docs` on UNIX and `%`*`ORACLE_HOME`*`%webcache\docs` on Windows.
2. Specify the file name of the apology page in the Apology Page (General Configuration > Apology Pages) of the Oracle9*i*AS Web Cache Manager administrative interface.

Because ESI has its own language elements for exceptions, there is no default apology page for `<esi:include>` errors. If you plan to use `<esi:include>` tags for partial-page caching and you do not implement the `onerror` attribute or the `try|attempt|except` block, then you must create an apology page. The onerror attribute is used before the `try|attempt|except` block. If the `try|attempt|except` block does not exist, then the exception handling is propagated to the parent or template page. The parent page will use the apology page, `onerror` attribute, or `try|attempt|except` block to handle the error. (Note that for the Oracle9*i*AS Web Cache 2.0.0.x and 9.0.2.x releases, when an apology page is configured, Oracle9*i*AS Web Cache bypasses any ESI programmatic exception handling code as described in bug 2412543.)

To configure exception handling:

1. Try to use the ESI `onerror` attribute or the `try|attempt|except` block for exception handling.
2. When it is not possible to use ESI language elements for exception handling, create an apology page in `$`*`ORACLE_HOME`*`/webcache/docs` on UNIX and`%`*`ORACLE_HOME`*`%\webcache\docs` on Windows: If there are many ESI fragments and you do not want display errors for each fragment, then configure Oracle9*i*AS Web Cache to serve a one-byte blank apology page for ESI errors. If you want to display a generic error message for each fragment that fails, then configure Oracle9*i*AS Web Cache to serve an apology page containing the error message.
3. Specify the file name of the apology page in the Apology Page (General Configuration > Apology Pages) section of the Oracle9*i*AS Web Cache Manager administration interface.

5.19 Use Redirection to Cache Entry Pages

For some popular site entry pages, such as "/", that typically require session establishment, session establishment effectively makes the page non-cacheable to all new users without a session. To cache these pages while preserving session establishment, you can either:

- Create a blank page that provides session establishment for all initial requests and redirects to the real popular page. This way, subsequent redirected requests to the popular page will carry the session, enabling the popular page to be served out of the cache.
- Use a JavaScript that sets a session cookie for the popular pages.

For more information on configuring caching rules for pages requiring session

establishment, see Chapter 7, “Creating Caching Rules,” of the *Oracle9iAS Web Cache Administration and Deployment Guide*.

5.20 Use the <esi:environment> Tag for Authentication/Authorization Callbacks

Some applications protect certain Web pages with authentication or authorization or validate session information in the HTTP request. Even though the page content can be cached, every HTTP request must be authenticated, authorized, or validated by the origin server. For these pages, it is not appropriate to cache the full page. While it is possible to utilize JavaScript to achieve authentication, authorization, and validation through a separate HTTP request, the `<esi:environment>` tag provides a better solution to this problem.

An ESI environment is a type of fragment with a response that defines a set of variables that can be accessed by response variable occurrences in the enclosing template. The tag itself does not contribute to the final assembled output. If a page has cacheable content but requires mandatory authentication, authorization, and session validation, you can enclose an `<esi:environment>` tag in the page referencing a non-cacheable environment, and cache the enclosing page as desired. When the cached page is requested, an HTTP request that specifies the environment will always be sent to the origin server, making a callback to the application. In this callback request, if you want to validate a cookie in the enclosing page request for session validation, authorization, or authentication, then specify the `<esi:environment>` tag to include that cookie in its request. It is also possible to include other information from the page request in the environment request.

If authentication, authorization, and validation are passed, your application should return HTTP status code 200 and any ESI environment response. Oracle9*i*AS Web Cache will proceed to finish assembling the page. If the authentication, authorization, and validation fail, then your application should return an appropriate HTTP status code at or above 500 to denote a server error, or between 400 and 499 to denote a client request error. Oracle9*i*AS Web Cache will then recognize that this environment has failed, and resort to standard ESI exception handling to abort this page or output an alternative error page or login page.

For more information about using the `<esi:environment>` tag or implementing ESI exception handling, see Appendix D, “Edge Side Includes Language,” of the *Oracle9iAS Web Cache Administration and Deployment Guide*.

5.21 Use Cookies and URL Parameters to Increase Cache Hit Ratios

Oracle9*i*AS Web Cache can cache different versions of a document with the same URL based on request cookies or headers. To use this feature, applications may need to implement some simple change, such as creating a cookie or header that differentiates the pages.

On the opposite side of the spectrum, some applications contain some insignificant URL parameters that lead to different URLs representing essentially the same content. If the documents are cached under their full URLs, then the cache-hit ratio becomes very low. You can configure Oracle9*i*AS Web Cache to ignore the non-differentiating URL parameter values when composing the “cache key” for documents, so a single document will be cached for different URLs, greatly increasing cache hit ratios.

Sometimes the content for a set of pages is nearly identical, but not exactly the same. For example, the pages may contain hyperlinks composed of the same URL parameters with different session-specific values, or they may include some personalized strings in the page text, such as welcome greetings and shopping cart totals. In this case, Oracle9*i*AS Web Cache can still store one single copy of the document with placeholders for the embedded URL parameters and/or the personalized strings, and dynamically substitute the correct values into the placeholders when serving the document to clients.

You can also control whether a cached document can be served to a client based on its session state.

For more information on multiple version documents, sessions, ignoring URL parameter values, simple personalization, and how to control whether a cached document can be served to a request based on sessions, see Chapter 2, “Caching Concepts,” of the *Oracle9iAS Web Cache Administration and Deployment Guide*.

5.22 Use a Network Load Balancer in Front of Oracle9*i*AS Web Cache

Many customers deploy a single instance of Oracle9*i*AS Web Cache in front of their application Web server farm. In such deployments, the Oracle9*i*AS Web Cache acts as the virtual IP address for the application, in addition to providing caching and load balancing services. This deployment is both functionally sufficient and cost-effective for customers that do not require 100 percent application uptime. The Oracle9*i*AS Web Cache is highly stable and, in the event of a failure, a process monitor will automatically restart the cache.

For customers who cannot tolerate a single point of failure, Oracle Corporation recommends that two or more nodes running Oracle9*i*AS Web Cache be deployed behind a third-party hardware load balancing device. In turn, customers should use the built-in load balancing functionality in Oracle9*i*AS Web Cache to distribute cache miss traffic over the application Web server farm. Please refer to Chapter 4, “Deploying Oracle9*i*AS Web Cache,” of the *Oracle9iAS Web Cache Administration and Deployment Guide* for more information.

5.23 Use Oracle9*i*AS Web Cache Load Balancing Functionality for Availability and Scalability of Origin Servers

Situated between browser clients and the origin servers, the Oracle9*i*AS Web Cache introduces a new tier into the traditional Web farm architecture. To reduce complexity and to avoid the cost of purchasing additional load balancing hardware, Oracle9*i*AS Web Cache includes built-in load balancing and failover detection features to ensure that cache misses are directed to the most available, highest performing origin server in the farm. The cache supports both stateless and stateful load balancing mechanisms, including the use of cookies and URL parameters to maintain server affinity when required. In addition, Oracle9*i*AS Web Cache maintains a pool of HTTP connections between the cache and the origin Web servers to reduce connection establishment overhead and improve cache miss performance.

To avoid a single point of failure, two or more nodes running Oracle9*i*AS Web Cache should be deployed behind a third-party hardware load-balancing device. However, Oracle Corporation also recommends that customers use the built-in load balancing and failure detection functionality in Oracle9*i*AS Web Cache to route cache miss requests to origin Web servers. Deploying additional load balancing hardware between the Oracle9*i*AS Web Cache and origin server tiers is not recommended for the following reasons:

- Cost: Using another tier of load balancing hardware adds significant cost to a deployment, in part because these devices must also be deployed in pairs for high availability reasons.
- Complexity: Another tier of load balancing hardware is another set of systems to configure, manage, and troubleshoot.
- Features: Oracle9*i*AS Web Cache includes patent-pending performance assurance and surge protection features that enable customers to sustain higher loads with less application and database server hardware. These features depend on the capacity-based load balancing algorithms in Oracle9*i*AS Web Cache.

For more information on load balancing, performance assurance and surge protection functionality, see the *Oracle9iAS Web Cache Administration and Deployment Guide* and various technical white papers on Oracle9*i*AS Web Cache

5.24 Improve Response Times and Reduce Network Bandwidth With Compression

Oracle9*i*AS Web Cache features automatic compression of dynamically generated content. On average, using the standard GZIP algorithm, Oracle9*i*AS Web Cache is able to compress text files, such as HTML and XML by a factor of four. Because compressed objects are smaller in size, they require less bandwidth to transmit and can be delivered faster to browsers. With compression, everyone benefits: Internet Service Providers (ISPs), Hosting Service Provider (HSPs), corporate networks and content providers reduce their transmission costs, while end-users enjoy more rapid response times. Since 1997, all major browsers support the expansion of GZIP encoded documents.

Most application Web servers on the market are capable of serving compressed pages, but few enable caching of compressed output. With Oracle9*i*AS Web Cache, compression is a simple Yes / No option that an administrator selects when specifying a caching rule. Because Oracle9*i*AS Web Cache supports regular expression for caching rules, compression can be applied to responses using criteria other than just file extension. Regular expression makes it very easy to select which pages to compress and which pages not to compress, as well as whether or not a particular browser should receive compressed content. Unlike the typical application Web server, Oracle9*i*AS Web Cache offers compression and caching for pages that have been dynamically generated. By caching compressed output, Oracle9*i*AS Web Cache reduces the processing burden on the application Web server, which would otherwise have to re-generate and compress dynamic pages each time they are requested. Because compressed objects are smaller in size, they are delivered faster to browsers with fewer round-trips, reducing overall latency. In addition, compressed objects consume less cache memory.

Do not compress images, such as GIFs and JPEGs, as well as executables and files that are already zipped with utilities like WinZip and GZIP. Compressing these files incurs additional overhead without the benefits of compression. Also, do not compress JavaScript includes (.js) and Cascading Style Sheets (.css), as some browsers have difficulty expanding these file types.

For more information on compression, see the *Oracle9iAS Web Cache Administration and Deployment Guide.*

5.25 Deploy Caches in Remote Offices for Faster Response Times and Reduced WAN Traffic

Oracle9*i*AS Web Cache offers hierarchical caching features that enable customers to easily create Content Delivery Networks (CDNs). For high availability and performance, many Internet businesses mirror their Web sites in strategic geographical locations. Caching is an excellent low-cost alternative to full-scale mirroring. Caching may also be used to serve local markets in order to shorten response times to these markets, and to reduce bandwidth and rack space costs for the content provider. Within the corporate intranet, so-called "Enterprise" CDNs (eCDNs) provide shorter response times for branch office users of e-business applications. Compared to application mirroring and database replication, eCDN is a more manageable and cost-effective model of distributed computing. Using Oracle9*i*AS Web Cache, customers can distribute the content assembly and delivery functions of their applications to key network access points, while maintaining centralized management of application logic and data.

In setting up an eCDN, customers typically deploy Oracle9*i*AS Web Cache in each branch office data center as well as in the central office where the application and database are maintained. For example, a U.S.-based company might deploy instances of Oracle9*i*AS Web Cache in its U.S., Japanese, and Australian offices. The central cache residing in the U.S. serves as the origin server for the caches in Japan and Australia. Using various commercially available DNS routing techniques, requests are handled by the cache that is closest to the end user. A browser request made by a Japanese employee, for instance, is handled by the cache instance in Japan, thereby reducing WAN traffic and eliminating long-haul network latencies. In a distributed cache hierarchy, the central cache is aware of the branch office caches. As a result, any content invalidation messages sent to the central cache automatically propagate to the remote caches. This invalidation propagation feature ensures content consistency across the CDN and simplifies the management of cache hierarchies.

5.26 Turn Off Verbose Logging to Conserve Resources

Verbose event logs are useful for debugging purposes. However, verbose event logging consumes system resources. Unless you need the verbose event log to diagnose problems, you should disable verbose mode. Oracle9*i*AS Web Cache will write only typical events to the event log.

For information on specifying these settings, see Chapter 8, “Administering Oracle9*i*AS Web Cache,” of the *Oracle9iAS Web Cache Administration and Deployment Guide*.

5.27 Use the Oracle9*i*AS Web Cache Manager to Avoid Configuration Problems

The Oracle9*i*AS Web Cache Manager is a graphical user interface for administering, configuring, and managing caches. Oracle Corporation recommends that you use it rather than manually editing the configuration files to make configuration changes.

In cache clusters, it is especially important to use Oracle9*i*AS Web Cache Manager to modify the configuration and to propagate it to the other cluster members. If you manually edit the configuration files, you may encounter problems. For example:

If you do not set the site name properly for each cluster member, identical objects could be cached in each cluster member, but the objects may be known by a different name in the different caches. For example, you could have the same object cached as:

```
mysite:7777:document1.html
myste:7777:document1.html
```

If the site names are not consistent, the objects with the variant site name will not be invalidated even if you use invalidation propagation.

5.28 Use Web Caching to Help Defend Against Denial-of-Service Attacks

Oracle9*i*AS Web Cache was designed from the ground up to provide high performance, reliability and scalability on low-cost commodity hardware. A single Oracle9*i*AS Web Cache instance can be configured to support thousands of concurrent inbound HTTP connections. The throughput (requests/second) that a single cache instance can sustain scales linearly with CPU speed. Additionally, Oracle9*i*AS Web Cache fully supports the HTTP/1.0 and HTTP/1.1 header fields, including `Keep-Alive. Keep-Alive` reduces the frequency of connection establishment and improves performance and scalability under heavy load.

When clustered, the capacity – the amount of content stored in RAM – of the cache tier scales linearly, as well. Cache clustering achieves high availability by failing over among cluster nodes, as well as high scalability by utilizing memory and processing power of multiple inexpensive computing hardware units in parallel.

Not surprisingly, many customers have reported the successful use of Oracle9*i*AS Web Cache to prevent distributed and single-source denial-of-service attacks. Denial-of-service attacks attempt to prevent access to Web sites either by flooding sites with traffic volumes that surpass throughput and connection capacities or by sending malicious requests intended to exploit software flaws that cause servers to crash. By caching responses to denial-of-service requests, Oracle9*i*AS Web Cache helps address the throughput and scalability limitations of Web sites, while creating a crucial barrier between malicious queries and a site's origin application and database servers.

The guidelines for configuring connection limits, caching rules, and cache clusters are outlined in the *Oracle9iAS Web Cache Administration and Deployment Guide*.

5.29 Tune Invalidation Performance Using Indexes

To improve performance of advanced invalidation requests that use `QUERYSTRING_PARAMETER` to match objects with the same embedded URL parameters, manually create an invalidation index. Because an invalidation index creates more depth to an internal invalidation tree used by Oracle9*i*AS Web Cache, Oracle9*i*AS Web Cache is able to categorize objects in the cache for faster lookup and traversal. To use this feature for a URL or a request `POST` body, consider moving the `URI` or `BODY` value to `QUERYSTRING_PARAMETER` instead.

To specify an invalidation index, add an `INVALIDATIONINDEX` element to the `webcache.xml` file that specifies the substring value used for `QUERYSTRING_PARAMETER`:

```
<SECURITY>
...
</SECURITY>
<INVALIDATIONINDEX>
    <INDEXPARAM VALUE="VALUE_of_QUERYSTRING_PARAMETER"/>
    <INDEXPARAM VALUE="VALUE_of_QUERYSTRING_PARAMETER"/>
</INVALIDATIONINDEX>
<WATCHDOG ENABLE="YES|NO"/>
```

Oracle Corporation recommends using invalidation indexes to create more depth to flat directory structures.

For more information about invalidation, see Chapter 8, "Administering Oracle9*i*AS Web Cache," of the *Oracle9iAS Web Cache Administration and Deployment Guide*.

5.30 Test Application Upgrades and Patches to Ensure Existing Cache and Session Rules Still Function Correctly

Though there is a growing trend to use options provided by Oracle9*i*AS Web Cache for specifying the caching rules dynamically with the Surrogate-Control response header, some sites will continue to use Oracle9*i*AS Web Cache Manager for configuring the rules statically. Typically this configuration is done at the start of the deployment cycle. After adequate testing in a staging area to validate the rules, Oracle9*i*AS Web Cache is deployed in a production environment. Problems arise when the backend application is upgraded for patches or with new versions and some or all of the earlier statically configured rules become not applicable and void. For example, if a site uses a session-related caching rule and, after applying a patch the name of the session cookie or session-embedded URL parameter changes, all the pages related to that rule will no longer be cacheable, resulting in poor performance for the site.

When applying application upgrades and patches, it is important to understand the extent of the application changes and then verify and tune the related cacheability rules in Oracle9*i*AS Web Cache. By periodically checking the cache-hit percentage and ensuring that it remains more or less constant, you can guard against unexpected behavior. Whenever there is a major change in the database or the mid-tier layer, such as for upgrades or application patches, you should validate cacheability rules much the same way as you did during the initial deployment cycle, including but not limited to using verbose event logging. And if possible, include Oracle9*i*AS Web Cache in your application regression test cycle.

5.31 Use HTTPS for Administration, Invalidation, and Statistics Monitoring

The default configuration for Oracle9*i*AS Web Cache does not enable HTTPS for administration, invalidation, or statistics monitoring requests. Instead, these ports are configured for HTTP basic authentication. On an insecure network, the passwords for the `administrator` user and the `invalidator` user can be decoded if they are sniffed out of the HTTP traffic. To avoid breach of security information for unprotected and insecure networks, set the communication protocol to HTTPS for administration and invalidation operations in the Operation Ports page (Cache-Specific Configuration > Operations Ports) of Oracle9*i*AS Web Cache Manager.

6
Oracle HTTP Server Best Practices

This chapter describes Oracle HTTP Server best practices. The topics include:

6.1 Configure Appropriately for Modem Connections

Most Web sites have visitors connecting from slow modems. Sometimes, it takes longer for data to transfer over these slow connections than for data to be computed by the application server. Thus, an Oracle HTTP Server process can be blocked doing the transfer, and CPU processing power is not available for another request to perform computation.

If this is perceived to be a problem in your environment, you should front-end Oracle HTTP Server with either (a) Oracle9*i*AS Web Cache, which has a threaded architecture, or, (b) Oracle HTTP Server in reverse proxy mode, which can spawn more light weight processes to handle the transfer. In both cases, the backend Oracle HTTP Server is reserved to do the computation work.

This separation of data computation and data transfer responsibilities buffers a site from latency due to slow modem connections.

6.2 Tune TCP/IP Parameters

Setting TCP/IP parameters can improve Oracle HTTP Server performance dramatically.

See the chapter titled “Optimizing Oracle HTTP Server” in the *Oracle9i Application Server Performance Guide* for a detailed explanation of each of these parameters.

6.3 Tune KeepAlive Directives

The `KeepAlive`, `KeepAliveTimeout`, and `MaxKeepAliveRequests` directives are used to control persistent connections. Persistent connections are supported in HTTP1.1 to allow a client to send multiple sequential requests through the same connection.

Setting `KeepAlive` to “On” allows Apache to keep the connection open for that client when the client requests it. This can improve performance, because the connection has to be set up only once. The trade-off is that the httpd server process cannot be used to service other requests until either the client disconnects, the connection times out (controlled by the `KeepAliveTimeout` directive), or the `MaxKeepAliveRequests` value has been reached.

You can change these `KeepAlive` parameters to meet your specific application needs, but you should not set the `MaxKeepAliveRequests` to 0. A value of 0 in this directive means there is no limit. The connection will be closed only when the client disconnects or times out.

You may also consider setting `KeepAlive` to “Off” if your application has a large population of clients who make infrequent requests.

6.4 Tune MaxClients Directive

The `MaxClients` directive controls the maximum number of clients who can connect to the server simultaneously. This value is set to 1024 by default.

If your requests have a short response time, you may be able to improve performance by setting `MaxClients` to a lower value. However, when this value is reached, no additional processes will be created, causing other requests to fail.

In general, increasing the value of `MaxClients` does not improve performance when the system is saturated.

If you are using persistent connections, you may require more concurrent httpd server processes, and you may need to set the `MaxClients` directive to a higher value. You should tune this directive according to the `KeepAlive` parameters.

6.5 Avoid Any DNS Lookup

Any DNS lookup can affect Apache performance. The `HostNameLookups` directive in Apache informs Apache whether it should log information based on the IP address (if the directive is set to “Off”), or look up the hostname associated with the IP address of each request in the DNS system on the Internet (if the directive is set to “On”).

Oracle has found that performance degraded by a minimum of about 3% in our tests with `HostNameLookups` set to on. Depending on the server load and the network connectivity to your DNS server, the performance cost of the DNS lookup could be much higher. Unless you really need to have host names in your logs in real time, it is best to log IP addresses and resolve IP addresses to host names off-line.

6.6 Turn Off Access Login if You Do Not Need to Keep an Access Log

It is generally useful to have access logs for your Web server, both for load tracking and for the detection of security violations. However, if you find that you don’t need these data, you should turn it off and reduce the overhead of writing the data to this log file.

6.7 Use FollowSymLinks and Not SymLinksIfOwnerMatch

The `FollowSymLinks` and `SymLinksIfOwnerMatch` options are used by Apache to determine if it should follow symbolic links. If the `SymLinksIfOwnerMatch` option is used, Apache will check the symbolic link and make sure the ownership is the same as that of the server.

6.8 Set AllowOverride to None

If the `AllowOverride` directive is not set to None, Apache will check for directives in the `.htaccess` files at each directory level until the requested resource is found for each URL request. This can be extremely expensive.

6.9 Use mod_rewrite to Hide URL Changes for End Users

Oracle9*i*AS includes a component that can transparently map the URLs visible to the end users to a different URL – without requiring a round-trip to the browser, or, any code change.

This feature makes it very easy to re-organize directories on the server side, or, other changes even after an application has been developed and deployed. There is a slight performance impact, however, as this configuration change for mod_rewrite is generally preferred over unfriendly URLs.

7

Oracle9*i*AS Portal Best Practices

This chapter describes Oracle9*i*AS Portal best practices. It features the following sections:

- Performance Best Practices
- Content Management and Publishing Best Practices
- Best Practices for Oracle9iAS Portal Export/Import

In addition to the following, more Oracle9*i*AS Portal best practices can be found at:

`http://portalcenter.oracle.com`

On that Web page, click on Product Information

7.1 Performance Best Practices

This section describes performance best practices. It features the following topics:

- Use Appropriate Caching Strategy Depending on Page Content
- Use Web and Database Providers Judiciously
- Improve Availability and Scalability by Having Multiple OC4J_Portal Instances
- Scale Oracle9iAS Portal Better by Tuning Oracle9iAS Infrastructure Database Optimally
- mod_plsql Tuning Directly Impacts Oracle9iAS Portal Performance
- Leverage Web Provider Session Caching
- Increase Execution Speed of Slowest Portlet to Increase Page Assembly Speed
- Reduce Page Complexity to Improve Cachability
- Measure Tuning Effectiveness Regularly to Improve Performance

7.1.1 Use Appropriate Caching Strategy Depending on Page Content

Oracle9*i*AS Portal provides two different mechanisms of caching to improve performance. These include out-of-the-box integration with Oracle's award winning in-memory cache solution, Oracle9*i*AS Web Cache, and a persistent file-based cache.

The file-based cache is always maintained at the same time as Web cache, providing redundant cover should the in-memory solution fail. It is also useful for things like persistent sessions. In the future, persistent content may be used for cache warming of Web cache. Further, file-cache performance improvements can be achieved by locating the cache on a RAM Disk.

By default, Oracle9*i*AS Portal issues dynamic caching instructions to Oracle9*i*AS Web Cache, to allow caching of default content (e.g. page). The page designer can use the different cache options (whole page, page definition, or none) to ensure that the correct balance is maintained between speedy delivery of cached content and avoiding the delivery of stale content. It is, however, important that the page or portlet designer understand that the degree of dynamism of a web page is inversely proportional to its cachability. Understanding validation, expiration, and invalidation-based caching will help the designer select the most appropriate cache method.

For superior performance, Oracle9*i*AS Web Cache should be deployed on a dedicated machine - by default, it is installed, configured, and co-located with the Oracle9*i*AS Portal middle-tier. Review the Oracle9*i*AS Web Cache best practices for further details.

7.1.2 Use Web and Database Providers Judiciously

There are two different types of providers: web provider and database provider. Database providers are implemented in Java or PL/SQL and executed as stored procedures within the Oracle database. The Oracle9*i*AS Portal middle-tier communicates with these providers in two ways: through mod_plsql, if the provider resides in the local portal database; or by SOAP over HTTP, if the provider resides in a remote database. (In Oracle9*i*AS Portal 3.0.x, Net8 connections were used in both cases. SOAP over HTTP allows easier communication with remote database providers through firewalls.) Note that execution in the database does not place any restrictions on the functionality of a portlet; database facilities allow for external communication in many ways, including HTTP connections to external content. Database providers are particularly appropriate for portlets that require significant interaction with the database and in situations where the development team has extensive Oracle PL/SQL development experience.

Web providers are implemented in any Web deployment environment (e.g. Java, ASP, Perl) and executed as an application external to Oracle9*i*AS Portal. Oracle9*i*AS Portal also communicates with these providers using SOAP over HTTP. Web providers are most appropriate for external information sources (e.g. Internet news/business information) and in environments where developers have experience using Java and other Web development languages.

7.1.3 Improve Availability and Scalability by Having Multiple OC4J_Portal Instances

Oracle9*i*AS Portal provides a parallel page engine (PPE) stateless servlet that fetches the page meta data, assembles the page, and manages the cache. This is a key component, and its stateless nature allows it to be deployed across multiple OC4J instances.

By default, Oracle9*i*AS is installed with a single OC4J_Portal instance where this PPE servlet is deployed. From a scalability perspective, it is highly recommended to have at least one other (if not more) OC4J instance with this PPE servlet deployed. Alternatively, you can increase the number of OC4J processes dedicated to this instance. Oracle HTTP Server load balance routing will distribute requests across the multiple instances or processes, providing better scalability for the system.

7.1.4 Scale Oracle9*i*AS Portal Better by Tuning Oracle9*i*AS Infrastructure Database Optimally

Oracle9*i*AS Infrastructure provides important functionality to Oracle9*i*AS Portal: all the metadata, database providers, etc. reside in the database. Hence, conventional database tuning (e.g. putting portal Indexes on a separate disk) is important. However, it is not recommended to analyze the schema for further fine tuning since this has already been done and CBO (Cost Based Optimizer) used where appropriate.

Moreover, there are also standard ongoing jobs that re-tune the schema based on collected statistics on a regular basis.

Another aspect of tuning is the SQL*NET tuning between the mod_plsql (i.e. Oracle HTTP Server) machine and the database itself. You should also consider Real Application Clusters (RAC) as an option for the availability and scalability of the Oracle9*i*AS Portal database.

7.1.5 mod_plsql Tuning Directly Impacts Oracle9*i*AS Portal Performance

mod_plsql maintains it's own connection pool thus negating the need in most cases for the use of MTS. There are, however, some tuning parameters that can be adjusted to give optimal mod_plsql performance and ensure that processes are not shut down heavily:

- To reduce latch contention, ensure that the DAD setting for Session Management Type is set to `STATELESS_FAST_RESET`.
- Set `MaxClients=MaxSpareServers=average system` load.
- Set `MaxRequestPerchild=10000`
- Set `MinSpareServer=1`
- Set `KeepAlive` to off for heavily loaded sites.

Tuning these parameters will affect the performance of Oracle HTTP Server with respect to nonmod_plsql requests. Therefore, if you wish to service other types of requests with the same installation, you can employ a dual listener strategy where you start two Apache listeners, one tuned for standard content requests and one tuned for mod_plsql content requests. Further information about this approach and other factors can be read in the mod_plsql section of the *Oracle9i Application Server Performance Guide*.

7.1.6 Leverage Web Provider Session Caching

When registering a web provider, it is possible (by selecting a checkbox) to request that session specific information such as session id and user login time be cached for each request. Whilst this is a mandatory requirement for web providers who rely on session information to ensure the validity of atomic transactions, those providers who do not rely on session level information should turn this option off as it improves the portlet cache hit rate.

7.1.7 Increase Execution Speed of Slowest Portlet to Increase Page Assembly Speed

Portlet execution speed is the average time required to execute all uncached portlets on a page. Since portlets execute in parallel, this measure will be equal to the execution speed of the slowest portlet, plus page assembly overhead.

The portlet execution speed will differ from site to site since each site will have a different mix of content and applications in their portal. Estimating this number will require running your own benchmark tests (if you already have applications that you plan to expose through portlets, you will already have a sense of the execution times). In general, the speed of page assembly will be limited by the execution speed of the slowest portlet on the page (when the portlet output is not cached).

7.1.8 Reduce Page Complexity to Improve Cachability

Page security and the number of tabs and portlets on a page will affect the time it takes to generate page metadata. The number of portlets on a page will affect page assembly times in the middle tier especially if each portlet must be contacted for a validity check or content refresh.

Page complexity, a function of page security and the number of tabs, items, and portlets on a page, affects throughput by increasing the amount of metadata that needs to be generated as well as the number of security and validity checks. Page complexity does not affect page assembly time in the middle-tier but may affect the time it takes to validate and refresh portlet content.

7.1.9 Measure Tuning Effectiveness Regularly to Improve Performance

One of the ways to evaluate whether the practices mentioned here or elsewhere are effective is to measure the resulting performance, and use the measurements to further fine tune the system.

To get granular results from system internals, `append &_DEBUG=1` to the end of the portal page URL you wish to measure performance for. The resultant output is a report from the parallel page engine. It will provide details of performance of each component on the page, whether a cache miss or hit occurred and how long page loading took.

Periodic monitoring of this information will help keep the system fine-tuned for better performance.

7.2 Content Management and Publishing Best Practices

This section describes content management and publishing best practices. It features the following topics:

- Use a Single Page Group for Unrestricted Copying, Multiple Page Groups for Delegating Administration
- Research Taxonomy Before Committing to the Pages
- Always Use Page Templates Instead of Directly Creating Pages
- Separate Template Content From Layout
- Understand the Difference Between Attributes, Categories, and Perspectives
- Avoid Using Simple Item Types

7.2.1 Use a Single Page Group for Unrestricted Copying, Multiple Page Groups for Delegating Administration

Oracle9*i*AS Portal allows portal pages to be organized within page groups. The default and easiest way to get started is with a single page group. This has the advantage of being able to easily copy or move content elements across pages within the group.

However, in a larger environment, you may want different people to administer different areas of the site - and this delegation of administration is easier if you separate your site into multiple page groups. Currently, it is not possible to copy or move content elements between page groups, or to use templates, styles, and metadata elements (categories, perspectives, attributes, and item types) owned by one page group in another page group.

Templates, styles, and metadata that need to be used across page groups can be placed in the supplied Shared Object Page Group. In addition, any page can be published as a portlet, which allows the content on that page to be viewed on any other page in any page group.

7.2.2 Research Taxonomy Before Committing to the Pages

There are several different ways to organize the set of content that your portal needs to provide to its end users. This organization is referred to as a taxonomy.

You can create a physical taxonomy consisting of pages and sub-pages. You can also create virtual taxonomies consisting of categories and perspectives. Users can browse a taxonomy, which appears as a hierarchy of pages. Each category and perspective page is built dynamically by searching for content belonging to the selected category or perspective when the page is rendered.

Reorganizing the physical taxonomy is easy and can be done by moving pages around within a page group and by moving items between pages (however, pages and items cannot be moved between page groups, as described above). Currently, it is not possible to reorganize categories and perspectives. While content can be reassigned to a different category or perspective, this can be a cumbersome process if you have a large number of items or pages.

Therefore, carefully plan your category and perspective hierarchies before you start to add content to your pages.

7.2.3 Always Use Page Templates Instead of Directly Creating Pages

Creating pages with Oracle9*i*AS Portal is easy, and it may be tempting to start adding pages quickly at the onset of your portal development project without first defining page templates. To ensure a consistent look-and-feel to your site and to minimize maintenance effort, it is recommended that you always base your pages on page templates. A template cannot be applied to a page once the page has been created.

It is also a good idea to manage your templates in the Shared Objects page group, as templates cannot be promoted from another page group to Shared Objects.

This also implies that you should define your templates before you create your content page groups, as you will want the root page for your page group to use one of your standard templates. The root page of a page group is automatically created, and you cannot replace the root page with another page.

7.2.4 Separate Template Content From Layout

A page template defines the layout (the placement of item and portlet regions) for pages. A template can also contain content, in the form of portlets and items that you want to appear on all its pages. However, changes made to a template are seen immediately in its dependent pages. Making changes to the content on a template will take some period of time (minutes or even hours, depending on the extent of the changes). During this time, the pages themselves will be in a state of flux as the template is modified. This can have an undesirable impact on your portal users and may require that the affected pages be unavailable while performing template maintenance.

This situation can be avoided by managing template content on navigation pages. For example, use a navigation page to contain the banner for your template, including such elements as the page name smart text, company logo, and various smart links such as the "Customize" icon and the "Home Page" link. When you want to modify the banner, copy the navigation page and make changes to the copy. When you are satisfied with the changes, replace the original banner navigation page portlet on the template with the modified copy. This can be done very quickly and will have minimal impact on portal users. The same recommendation applies to navigation bars, page footers, and other content that you want to include on the template.

7.2.5 Understand the Difference Between Attributes, Categories, and Perspectives

One of the big advantages of Oracle9*i*AS Portal is the ability to define and associate metadata with any content. Oracle9*i*AS Portal provides three types of metadata:

1. Categories – "what is it?"
2. Perspectives – "who is the audience?"
3. Attributes for any other descriptive information.

It is important to understand the characteristics of these metadata elements in terms of their impact on content organization, maintenance, presentation, and search. For example, they all aid in searching for content but have different styles for search submission and for presentation in search results. There are also important differences in terms of how content contributors assign metadata values to content and in how these elements are presented on pages.

The following Table 7–1 summarizes the characteristics of the metadata elements:

Table 7–1 Metadata Elements

Characteristic	Custom Attribute	Categories	Perspectives
Can be mandatory on Add/Edit item.	Yes	Yes	No
Can be selected for Group By in Region display.	No	Yes	No
Can be arranged in a navigable hierarchy	No	Yes	Yes
Allows multiple values for a single item.	No	No	Yes
Select values from Static List of Values	Yes	Yes	Yes
Select values from Dynamic List of Values (based on SQL Query)	Yes	No	No
Can be associated with an icon.	No	Yes	Yes
Searchable	Yes	Yes	Yes
Can be shown in Item display	Yes	Yes	Yes
Can be shown in Search Results	Yes	Yes	Yes

Table 7–1 Metadata Elements(Cont.)

Characteristic	Custom Attribute	Categories	Perspectives
Can be used to order a custom query (using SQL against the `WWSBR_ALL_ITEMS` repository view)	No	Yes	No
Translatable	Yes	Yes	Yes
Data Types	Boolean, Date, File, Number, PL/SQL, Text (Single or Multi-Line), URL	Text Only	Text Only

7.2.6 Avoid Using Simple Item Types

Simple Item Types (e.g. Simple Text, Simple File, Simple URL, Simple Image) are very basic item types that contain a minimum number of attributes (the Display Name and the item type's distinguishing attribute, such as the text box for Simple Text and the file upload box for Simple File). Custom item types (including the pre-defined types Text, File, URL, Image, etc.) are based on the simple types but can contain any number of additional attributes and can be modified anytime to add or remove attributes.

While Simple Item Types present a very simple interface for end users to add content, it is important to understand that they cannot be extended to include additional attributes, and you cannot change the type of an item once that item has been created. Therefore, if you anticipate that you will want to add additional attributes to your content, avoid the use of simple item types by following these steps:

1. If you want to present a simple interface for adding items, create new item types that look like the simple types.
2. Configure your page groups to hide the Simple Item Types and to make only custom types visible.

Note that items created through the WebDAV interface (Web folders) are always created as Simple File. While WebDAV makes it very convenient to add content from the desktop, that convenience may be offset by the restriction on adding attributes to the Simple File item type. If you need the flexibility associated with a custom item type, create your file items through the Add Item wizard in the browser. Any file item can be edited through WebDAV, even if it belongs to a custom item type. In a future release of Oracle9*i*AS Portal, you will be able to specify a custom type for items created through WebDAV.

7.3 Best Practices for Oracle9*i*AS Portal Export/Import

Oracle9*i*AS Portal provides a set of export/import utilities which enable customers to copy portal content between portal installations. A typical example where these utilities would be used is to copy or update portal page groups and application components between a development instance and a production instance of Oracle9*i*AS Portal.

The following is a summary of recommendations and best practices developed for Export/Import functionality as provided in Oracle9*i*AS Portal version 9.0.2.2:

- General Guideline/Best Practices for Oracle9iAS Portal 9.0.2.2 Export/Import
- Best Practices System Checklist Before Performing a Portal Export/Import Operation
- Best Practices for Building Transport Sets
- Best Practices For Configuring Your Portal Content For Maximal Portability for Export/Import Operations
- Best Practices for Exporting/Importing Page Groups and Components
- Best Practices for Exporting/Importing Web Providers
- Best Practices for Exporting/Importing Users and Groups
- Best Practices for Troubleshooting Oracle9iAS Portal 9.0.2.2 Export/Import

7.3.1 General Guideline/Best Practices for Oracle9*i*AS Portal 9.0.2.2 Export/Import

If you are not sharing the same Oracle Internet Directory instance between your source and target systems, Oracle recommends the following procedure for export/import:

- Develop your portal objects (page groups, content, applications, etc.) on your source/development system.
- To simplify the task of export/import, assign users, groups and privileges ONLY on your production system.
- Use Export/Import to promote your portal objects to your target/production system.
- Apply users and privileges to imported portal objects as needed.

If however, you are able to share a single Oracle Internet Directory instance between your source and target systems, we would recommend the following procedure:

- Develop your portal objects (page groups, content, applications, etc.) on your source/development system.
- Assign users, groups and privileges from your source system.
- Use Export/Import with "Export Security" checkbox enabled to promote your portal objects and privileges to your target/production system.

7.3.2 Best Practices System Checklist Before Performing a Portal Export/Import Operation

Before exporting or importing, ensure that your system meets the minimum system requirements. You will also need to know the following information about your configuration on both your export and import portal environments:

- Portal Schema Name
- Portal Schema Password
- Portal Connect String Information
- Portal User Name
- Portal User Password
- Company Name (used only for hosted portal installations) - leave blank in most cases.

Also, plan to perform the export and import process during non-business hours, and to disable access to Oracle9*i*AS Portal during the process. To disable access to portal temporarily for all other users, one way is to configure your listener for a different port number during the duration of the export and revert it back to the original port when your export is complete.

Each export or import process sets up a background process. Therefore, verify that the job_queue_processes database parameter is set appropriately.

To check the value of the job_queue_processes parameter, simply perform the following query from SQL*Plus:

```
select name, value from v$parameter where name='job_queue_processes'
```

The value for `job_queue_processes` should be at least 2 to allow toe execution of background jobs.

An alternative way of checking the `job_queue_processes` parameter is to simply examine the init.ora file in your database's *`ORACLE_HOME`*.

Also, for Oracle9*i*AS Portal version 9.0.2.2, please be sure to download the latest patchset for Export/Import from Metalink. The current patchset as of this writing is patchset 2617359.

7.3.3 Best Practices for Building Transport Sets

Once the minimum requirements are verified, your goal is to create a transport set. Transport sets contain the portal objects that you are planning to export to your destination portal environment. The steps below outline the basic process of creating a transport set for export. Please refer to the online documentation for the precise series of steps.

- Mark objects for export (from the Navigator, or search results > Bulk Actions for page groups).
- Select either: **Export this Transport Set Immediately** or **Export Later**, more objects have to be added.
- When all desired objects have been added to your transport set, select Export This Transport Set Immediately.
- Check the log in your transport set manager for any errors.
- Choose an appropriate export script based on your Operating system, and save the resulting file to the location where you will want to run the export utility (generally this location should be where your export portal resides).
- Run the script using -mode export as the option. This script will create a dump file that you will name.
- Using FTP, transfer your dump file and export/import script to the machine where your destination Oracle9*i*AS Portal repository resides.

7.3.4 Best Practices For Configuring Your Portal Content For Maximal Portability for Export/Import Operations

Before exporting/importing, please review Section 4 of the Oracle9*i*AS Portal 9.0.2 release notes, summarized here below:

1. **User and group privileges for given objects are not exported by default:**: For information on exporting users and groups refer to Section 7.3.1, "General Guideline/Best Practices for Oracle9iAS Portal 9.0.2.2 Export/Import" and Section 7.3.7.1, "Export/Import of Objects (With Security) Between Portals Using Different Oracle Internet Directory Servers in 9.0.2".
2. **Web providers**: Web Provider registration details are migrated, however, if a web provider with the same name exists in the target, that provider is reused. If one does not exist, then a new web provider is created.
3. **Portal export/import does not provide support for external and partner applications, and password stores**: Any object from these placed on a page may not work as expected when the page is migrated from one portal to another. For example, if you have an external applications portlet imported on a portal page from your source system, it will display the external applications native to the target system. As another example, if you have other external applications as portlets, and are having issues on your target system, you may want to remove the external application portlet first from your source system, import the page, and then add it back.
4. **Custom search is reset upon import.** These customizations will have to be re-entered.
5. **Custom Item Types -** Portal Export/Import of pages overwrites all attribute values of items based on custom item types with custom attributes in the target portal. For this reason, you should create items (based on custom item types) on the source portal only, and migrate them to the target through the export-import process for that page.
6. **Item URL behavior**: Item URLs are likely to change after import. To minimize any potential problems, be sure to follow these important guidelines on how to refer to image items via URL.

7. **Page URL behavior**: Always use page link item types or direct access URLs when creating links to portal pages. Do NOT use "raw" portal page URLs.

By default, portal page URLs generated by Oracle9*i*AS Portal contain installation specific ID numbers that change when the object is exported. This causes broken links when pages are imported into a different site.

Here is an example of a URL generated for a page. If the page is imported on another site, this PAGEID will change.

```
http://my.portal.com/servlet/page?_pageid=47,49&_dad=portalr2&_schema=portal
```

If you are using URLs such as the above as manually-entered links, we recommend instead the use of Direct Access URLs or Page Link item types instead.

The same page has this direct access URL:

```
http://my.portal.com/pls/portal/url/PAGE/HRPAGEGROUP/HRHOME/HRBENEFITS
```

To find the direct access URL for a page, look at the page property sheet. A link to the property sheet can be displayed by adding a Property Sheet Smart Link item to the page.

Also, you can also use a Page Link item type to create a link to a page. The Page Link item type dynamically generates the correct link at runtime.

To ensure that links do not 'break' when pages are imported into a different site, *always* use Page Link item types or direct access URLs when creating links to pages.

For more information, refer to 'Direct Access URLs' and 'Page Links' in the Oracle9*i*AS Portal online help.

7.3.5 Best Practices for Exporting/Importing Page Groups and Components

Page groups and their associated components may be moved from development to production via the export/import utilities described in this document. In addition to page groups as a whole, individual components within page groups such as subpages, categories, perspectives, page styles can be migrated individually to the import system, provided that the entire page group has been imported to the import system earlier.

Some considerations and best practices to keep in mind are the following:

- The first export to your target system should migrate the entire page group from the source portal to the target portal instance. Subsequent transport sets can then export an individual page, or other page group component on the destination portal installation, all new or existing content on a page will be overwritten when a page of the same name is being re-imported to the destination from elsewhere.
- You can only migrate objects within a page group to the same page group of the same name on the destination portal.
- After an initial import operation to your target system, if you change the name of the page group on the target system, subsequent import attempts to that page group will fail.

7.3.6 Best Practices for Exporting/Importing Web Providers

In the current Oracle9*i*AS Portal 9.0.2.2 release, your import operation will fail with a MERGE_FAILED error if your web providers cannot be reached or cannot be registered successfully during import.

Before importing on your target system, all web providers referenced by your transport set must either exist already or be able to be registered successfully during the import on your target system.To ensure successful registration, ensure that your web providers meet the following conditions on your target system:

- Ensure that you have connection to your web providers during the import operation. Your web providers must be up during the import operation.
- If you are using web proxies on either your import or export portal installations, ensure that your proxies are configured correctly on your import installation before importing.
- An alternative to keep in mind is to consider registering your web providers manually in advance of performing your import on the target system to help ensure that your import operation goes smoothly.
- If you register your web providers manually, they need to be the same name as the corresponding web provider(s) on your export system.
- If you have portal pages that reference database providers (application components), be sure also to import your database providers (application components) either first, or concurrently in the same transport set with your page groups.
- Another alternative is to remove offending web provider portlets (for those web providers that may not be contactable / available during the import operation) from pages before exporting, and then to add them back in after importing.

7.3.7 Best Practices for Exporting/Importing Users and Groups

As with the export/import of portal content, the need to export users and groups depends primarily upon whether your Oracle9*i*AS Portal instances are sharing the same Oracle Internet Directory instance or not.

Review the following topics:

- Export/Import of Objects (With Security) Between Portals Using Different Oracle Internet Directory Servers in 9.0.2

7.3.7.1 Export/Import of Objects (With Security) Between Portals Using Different Oracle Internet Directory Servers in 9.0.2

1. Use Oracle Internet Directory utilities to migrate Users/Groups from source Oracle Internet Directory server to target Oracle Internet Directory server.
2. In Oracle9*i*AS Portal, migrate portal objects with "export security" enabled from source portal to target portal.
3. (MANUAL) For the imported users, set the guid to null (`WWSEC_PERSON$.GUID`)
4. (MANUAL) For the imported groups, update the guid using the instructions found in the section below on how to update the GUID.

Please refer to the Oracle Internet Directory documentation on how to migrate users and groups between Oracle Internet Directory servers.

At the end of the 4 step process, you should be able to edit the user or group profile and also see them in the imported objects access list. Keep in mind the order of the process outlined above. All the steps are required and must be carried out sequentially.

7.3.7.1.1 How to update the GUID

First of all you have to use the following steps to determine whether or not the specific group you want to synchronize is a local group:

By default, all local groups are created within the portal group install base. You can find the group install base by running the following commands in the portal schema from SQL*Plus:

```
set serveroutput on
exec dbms_output.put_line(wwsec_oid.get_group_install_base)
```

If the tail of the group's distinguished name matches the value thus returned then the group is within the group install base else it is not. For instance, if the group install base is " `cn=portal_groups,cn=groups,o=oracle,dc=com` " and the group that you want to synchronize has a DN " `cn=Supervisors,cn=PORTAL_GROUPS,cn=Groups,o=oracle,dc=com` ", then the tail is considered as matching and it is a local group. To confirm whether this is true or not you can try the following in the portal schema from SQL*Plus (please substitute the required group name for `<enter_group_name_here>`)

```
select guid, dn from wwsec_group$ where name = upper('<enter_group_name_
here>');
```

If you see a row with the GUID and DN values then it is a local group else it is not.

Depending on whether or not it is a group within the portal group install base in Oracle Internet Directory you can use the following steps to synchronize the GUID of a specific group:

1. **For local groups (groups within the portal group install base)**

   Use the following steps to get the GUID:

   - Run the following SQL*Plus command in the portal schema:

     ```
     set serveroutput on
     exec dbms_output.put_line(wwsec_oid.get_group_info(p_group_name =>
     '<enter_group_name_here>', p_check_local => false).guid)
     ```

     This will display the GUID for the group.

   Use the following steps to update the group entry in the portal schema

   - Pass the value of GUID to replace <enter_guid_here> in the next command:

     ```
     update wwsec_group$ set guid = '<enter_guid_here>' where name =
     upper('<enter_group_name_here>'); commit;
     ```

2. **For non-local groups (groups outside the portal group install base)**

   You have to know the Oracle Internet Directory host, port, the administrator's DN and password to run the following command. You also have to either know the Distinguished Name of the group or you should be able to determine by looking at a DN whether or not it pertains to the group that you are interested in. Use the following steps to get the GUID:

   - Run the following command to find out the GUID of the group:

     ```
     ldapsearch -h <enter_OID_host_name_here> -p <enter_OID_port_number_here>
     ```

```
-D <enter_OID_admin_DN_here> -w <enter_OID_admin_password_here> -s sub
-b "<enter_subscriber_DN_here>"
"(&(objectclass=groupOfUniqueNames)(cn=<enter_group_name_here>))"
orclguid
```

Example:

```
ldapsearch -h oid.oracle.com -p 389 -D cn=orcladmin -w welcome1 -s sub
-b "o=oracle,dc=com"
"(&(objectclass=groupOfUniqueNames)(cn=Supervisors))" orclguid
```

This will display a list of all groups under the specified subscriber whose name matches the specified group name. Every group will be displayed in two lines, the first line will have the DN of the group and the second line will have its GUID.

Get the DN and the GUID that you are interested in.

- **Use the following steps to update the group entry in the portal schema**.

  Pass the value of GUID to replace <enter_guid_here> and DN to replace <enter_DN_here> in the next command:

  ```
  update wwsec_group$ set name = '(<enter_guid_here>)', guid = '<enter_
  guid_here>' where dn = wwsec_oid.get_normalized_dn('<enter_DN_here>');
  commit;
  ```

  For example if the DN and GUID obtained from the previous step are

  "`cn=Supervisors,cn=OTHER_ GROUPS,cn=Groups,o=oracle,dc=com`" and the GUID is "`A729208AFBCA4C38E0340800208A8B00`" respectively then the above update query will become:

  ```
   update wwsec_group$ set name = '(A729208AFBCA4C38E0340800208A8B00)',
  guid = 'A729208AFBCA4C38E0340800208A8B00' where dn = wwsec_oid.get_
  normalized_dn('cn=Supervisors,cn=OTHER_
  GROUPS,cn=Groups,o=oracle,dc=com');
  commit;
  ```

These steps should allow you to synchronize a specific group in the portal schema. Repeat this process if there are multiple groups involved.

7.3.8 Best Practices for Troubleshooting Oracle9*i*AS Portal 9.0.2.2 Export/Import

As mentioned earlier, be sure to review the latest information and FAQs on Oracle9*i*AS Portal Export/Import located in the Export/Import Migration and Upgrade folder at the following URL:

```
http://portalcenter.oracle.com
```

Please be sure to look for FAQs and documentation specific to Oracle9*i*AS Portal 9.0.2.2, as procedures and best practices for subsequent and prior releases of Oracle9*i*AS Portal do vary.

8

Oracle9*i*AS Wireless Best Practices

This chapter describes Oracle9*i*AS Wireless best practices. The topics include:

- Development Best Practices
- Deployment Best Practices

8.1 Development Best Practices

This section describes development best practices. The topics include:

- Use Hosted Instance to Test Applications
- Download Up-to-Date Device Simulators for Testing Applications
- Use Oracle9iAS Wireless XML
- Use JSPs to Generate Oracle9iAS Wireless XML
- Use the HTTP Adapter Over a Custom Adaptor
- Use Hosted Wireless Web Services (Mobile Modules) for Rapid Development

8.1.1 Use Hosted Instance to Test Applications

Since it takes some effort to install the appropriate Oracle9*i*AS Wireless infrastructure to test your multi-channel applications for all mobile browsers, Short Message Service (SMS) devices and voice, a hosted instance is available on the Mobile Studio. The URL to the hosted instance is:

`http://studio.oraclemobile.com.`

8.1.2 Download Up-to-Date Device Simulators for Testing Applications

Some devices constantly change their presentation format (represented by corresponding stylesheets in Oracle9*i*AS Wireless). Hence, you should always download and test with the latest simulators for the devices you are developing for. You can generally go to the device manufacturers web site to download the simulators.

8.1.3 Use Oracle9*i*AS Wireless XML

If you are developing a multi-channel application, use Oracle9*i*AS Wireless XML. The value that Oracle9*i*AS Wireless XML adds is the ability to write an application once for all channels. You need not develop separate presentation code for each mobile device your application supports. Hence, development effort and cycles are reduced as the complexities of developing for each specific device browser is removed.

8.1.4 Use JSPs to Generate Oracle9*i*AS Wireless XML

For J2EE applications, we recommend that you output Oracle9*i*AS Wireless XML using JSPs. The presentation model parallels that of using JSPs to output HTML for HTML browsers. In the JSPs, replace HTML tags with Oracle9*i*AS Wireless XML. Your J2EE applications can then be delivered to any device including voice.

8.1.5 Use the HTTP Adapter Over a Custom Adaptor

The HTTP adapter allows you to separate the application logic from the application server, and use the application server as a smart browser. This method allows for easier application management, easier application server upgrades, and simpler porting.

8.1.6 Use Hosted Wireless Web Services (Mobile Modules) for Rapid Development

Oracle9*i*AS Wireless offers location and messaging web services (also called Mobile Modules) as added value to your wireless applications. These services location-enable and message-enable your applications without requiring you to write the implementation logic for them. It is possible to deploy these services internally in your own deployment of Oracle9*i*AS Wireless. However, Oracle offers hosted versions of these services, which your applications can connect to over the Internet. Using the hosted versions reduces your development and deployment cycles as you do not need to deploy and provision your own version of these services.

8.2 Deployment Best Practices

This section describes deployment best practices. It features the following topics:

- Deploy Own Wireless Infrastructure or Use Hosted Versions
- Deploy Your Applications in Phases
- Use SMS for Targeted Content in Specific Geographical Regions
- Use Oracle9iAS Web Cache with Oracle9iAS Wireless

8.2.1 Deploy Own Wireless Infrastructure or Use Hosted Versions

If you wish to have low initial costs, you can deploy your Oracle9*i*AS Wireless (middle tier) and applications in-house. Then, all the other infrastructures can be hosted by a service provider. Oracle provides wireless infrastructure as a hosted service.

8.2.2 Deploy Your Applications in Phases

The best method in deployment is to take a phased approach, where you release new channels in phases. Deploy channels based on your goals, for example, sales force PDA users who require your application urgently or a channel, which returns the quickest adoption rate or ROI. A common approach is to start with the PDA channel and then to voice.

8.2.3 Use SMS for Targeted Content in Specific Geographical Regions

SMS provides a way to make a simple request to get necessary information and is widely used in Europe and Asia. It is easy to get distracted with fancy images when developing for PDAs. SMS makes sure that only the vital content is made available, and the content is received in the most concise manner.

8.2.4 Use Oracle9*i*AS Web Cache with Oracle9*i*AS Wireless

Using Oracle9*i*AS Web Cache with Oracle9*i*AS Wireless can save bandwidth and server CPU cycles. The savings are in terms of device adaptation costs due to the fact that content can be shared across users and sessions. Additionally, applications are transformed only once (per device) from its multi-channel Oracle9*i*AS Wireless XML format.

9 Security Best Practices

This chapter describes security best practices. The topics include:

- General Best Practices
- OC4J Security Best Practices
- Oracle9iAS Single Sign-On Best Practices

9.1 General Best Practices

This section describes general best practices. The topics include:

- Best Practices for HTTPS Use
- Assign Lowest Level Privileges Adequate for the Task
- Best Practices for Cookie Security
- Best Practices in Systems Setup
- Best Practices for Certificates Use
- Follow "Common Sense" Firewall PracticesReview Code and Content Against Already Known Attacks
- Leverage Declarative Security
- Use the Oracle Integrated Version of JAAS
- Use Switched Connections in DMZ
- Place Application Server in the DMZ
- Tune the SSL SessionCacheTimeout Directive if You Are Using SSL

9.1.1 Best Practices for HTTPS Use

The following are recommended for using HTTPS with Oracle9*i*AS:

- **Configure Oracle9*i*AS to fail attempts to use weak encryption**. Oracle9*i*AS can be configured to use only specific encryption ciphers for HTTPS connections. Thus, connections from all old browsers that have not upgraded the client-side SSL libraries to 128-bit can be rejected. This ability is especially useful for banks and other financial institutions because it provides server-side control of the encryption strength for each connection.
- **Use HTTPS to HTTP appliances for accelerating HTTP over SSL.** You should in general use HTTPS everywhere you need to. However, the huge performance overhead of HTTPS forces a trade-off in some situations. Use of HTTPS to HTTP appliances can change throughput from 20/30 transactions per second on a 500MHz Unix to 6000 transactions per second for a relatively low cost, making this trade-off decision easier. Moreover, these are much better solutions than the math/crypto cards, which can be added to UNIX/NT/Linux boxes.
- **Ensure that sequential HTTPS transfers are requested through the same Web server.** Expect 40/50 milliseconds CPU time for initiating SSL sessions on a 500 MHz machine. Most of this CPU time is spent in the key exchange logic, where the bulk encryption key is exchanged. Caching the bulk encryption key will significantly reduce CPU overhead on subsequent accesses, provided that the accesses are routed to the same Web server. This improves performance.
- **Keep secure pages and pages not requiring security on separate servers.** While it may be easier to place all pages for an application on one HTTPS server, the resulting performance cost is very high. Reserve your HTTPS server for pages needing SSL, and put the pages not needing SSL on an HTTP server.

If secure pages are composed of many GIF, JPEG, or other files that would be displayed on the same screen, it is probably not worth the effort to segregate secure from non-secure static content. The SSL key exchange (a major consumer of CPU cycles) is likely to be called exactly once in any case, and the overhead of bulk encryption is not that high.

9.1.2 Assign Lowest Level Privileges Adequate for the Task

When assigning privileges to module(s), use the lowest levels adequate to perform the module(s) function(s). This is essentially "fault containment" which means if security is compromised, it is contained within a small area of the network and cannot invade the entire intranet.

9.1.3 Best Practices for Cookie Security

Use the following as guidelines for cookies:

- **Make sure that cookies have proper expiration dates.** Permanent cookies should have relatively short expiration dates of about three months or less. This will avoid cluttering client browsers, which may cause errors if the browser cannot transmit all the valid cookies. Non-permanent cookies should be set to expire when the relevant application exits.
- **Make sure that information in cookies should be "MAC'ed".** Method Authentication should be used to ensure that cookie data has not been changed since the application set the data. This helps ensure that the cookie cannot be modified and "trick" the application. Also, this helps prevent application failures if the cookie is inadvertently corrupted.
- **Make sure that the size and varieties of cookies are kept low.** There is a finite number and aggregate size of cookies that browsers support. If this is exceeded, then the browsers will not send all the relevant cookies leading to application failures. Also, very large cookies can result in performance degradation.
- **Carefully use cookie domain name facilities.** Use of cookie domains should ensure that the domain is the smallest possible. Making the domain oracle.com, for instance, would mean that ANY host in oracle.com would get the cookie. With hundreds of applications on different parts of oracle.com, a domain of oracle.com for each of them results in attempts to send hundreds of cookies for each HTTP input operation.

9.1.4 Best Practices in Systems Setup

Use the following as guidelines for system setup:

- **Apply all relevant security patches**. Check Metalink and TechNet for current security alerts. Many of these patches address publicly announced security holes.
- **When deploying software, change all default passwords and close accounts used for samples and examples.**
- **Remove unused services from all hosts**. Examples of unused services are FTP, SNMP, NFS, BOOTP, and NEWS. It is almost always worthwhile finding ways to eliminate FTP because it is especially noxious. HTTP or WebDAV may be good alternatives.
- **Limit the number of people with root and administrative privileges.**
- **In UNIX, disable the 'r' commands if you do not need them.** For example, rhost, rcp.

9.1.5 Best Practices for Certificates Use

Use the following guidelines when using certificates:

- **Ensure that certificate organization unit plus issuer fields uniquely identify the organization across the Internet.** One way to accomplish this would be to include the Dun and Bradstreet or IRS identification as identification for the issuer and the organizational unit within the certificate.
- **Ensure that certificate issuer plus distinguished name uniquely identify the user.** If the combination of issuer and distinguished name is used as identification, there is no duplication risk.
- **Include expiring certificates in tests of applications using certificates.** Expiration is an important consideration for a number of reasons. Unlike most username/password-based systems, certificates expire automatically. With longer duration certificates, fewer re-issues are required, but revocation lists become larger.

  In systems where certificates replace traditional usernames/passwords, expiring certificate situations may result in unexpected bugs. Careful consideration of the effects of expiration is required and new policies will have to be developed because most application and infrastructure developers have not worked in systems where authorization might change during transactions.

- **Use certificate re-issues to update certificate information.** Because certificates expire, infrastructure for updating expired certificates will be required. Take advantage of the re-issue to update organizational unit or other fields. In cases of mergers, acquisitions, or status changes of individual certificate holders, consider re-issuing even when the certificate has not yet expired. But pay attention to key management. If the certificate for a particular person is updated before it expires, for example, put the old certificate on the revocation list.
- **Audit certificate revocations.** Revocation audit trails can help you reconstruct the past when necessary. An important example is replay of a transaction to ensure the same results on the replay as during the original processing. If the certificate of a transaction participant was revoked between the original and the replay, failures may occur which would not have occurred when the original transaction was processed. For these cases, the audit trail should be viewed to simulated authentication at the time when the transaction was initially processed.

9.1.6 Review Code and Content Against Already Known Attacks

It is quite common for viruses or known attacks to resurface in slightly altered shape or form. Thus, just because a threat has been apparently eliminated does not mean it won't resurface. Use the following as guidelines to minimize the recurrence of the threat:

- **Ensure that programs are reviewed against double encoding attacks.** There area many cases where special characters, such as <, >, | are encoded to prevent cross-site scripting attacks or for other reasons. For example, '&lt;' might be substituted for '>'. In a double encoding, the attacker might encode the '&' so that later decoding might involve the inadvertent processing of a >, <, or | character as part of a script. Prevention of this attack, unfortunately, can only be provided by careful program review, although some utilities can be used to filter escape characters that might result in double encoding problems in later processing.
- **Ensure that programs are reviewed against buffer overflow for received data.**
- **Ensure that programs are reviewed against cross-site scripting attacks.** This attack typically tricks HTML and XML processing via input from browsers (or processes which act like browsers) to invoke scripting engines inappropriately. However, it is not limited to the Web technologies, and all code should be evaluated for this.

9.1.7 Follow "Common Sense" Firewall Practices

The following are some common recommended practices pertaining to firewalls.

While not unique to Oracle9*i*AS, these are important to overall Oracle9*i*AS security.

- Place servers providing Internet services behind an exterior firewall of the stateful inspection type. Stateful inspection means that the firewall keeps track of various sessions by protocol and ensures that illegal protocol transitions are disallowed through the firewall. This blocks the types of intrusion, which exploit illegal protocol transitions.
- Set exterior firewall rules to allow Internet-initiated traffic only through specific IP and PORT addresses where SMTP, POP3, IMAP, or HTTP services are running. Some protocols (e.g. IIOP) leave ports open with no receiving processes. PORT and IP combinations, which are not assigned to running programs, should not be permitted.
- Set interior firewall rules to allow messages through to the intranet only if they originate from servers residing on the perimeter network. All incoming messages must first be processed in the perimeter network.
- Send outgoing messages through proxies on the perimeter network.
- Do not store the information of record on bastion hosts. Information and processing should be segmented such that bastion hosts (fortified servers on the perimeter network) provide initial protocol server processing and generally do not contain information of a sensitive nature. The database of record and all sensitive processing should reside on the intranet.
- Disallow all traffic types unless specifically allowed. allow only the traffic required by Oracle9*i*AS(e.g. HTTP, AJP, OCI, LDAP) for better security.

9.1.8 Leverage Declarative Security

Oracle HTTP Server has several features that provide security to an application without requiring the application to be modified. These should be leveraged and / or evaluated before programming similar functionality as those features into the application. Specifically:

- Authentication - Oracle HTTP Server can authenticate users and pass the authenticated user-id to an application in a standard manner. (REMOTE_USER). It also supports single sign-on, thus reusing existing login mechanisms.

- Authorization - Oracle HTTP Server has directives that can allow access to your application only if the end user is authenticated and authorized. Again, no code change is required.
- Encryption - Oracle HTTP Server can provide transparent SSL communication to end customers without any code change on the application.

These three features should be leveraged heavily before designing any application specific security mechanisms.

9.1.9 Use the Oracle Integrated Version of JAAS

Oracle9*i*AS provides a J2EE compliant version of JAAS. One can use the infrastructure provided by Sun for configuring and managing users, roles and privileges or one can use JAAS integrated with Oracle9*i*AS Single Sign-On and Oracle Internet Directory infrastructure. The latter is better integrated with the rest of the Oracle infrastructure and results in less programming, better manageability, and more control than the standard offering. In addition, use of the Oracle infrastructure can greatly improve the scalability, failover, and management by substituting Oracle Internet Directory and other Oracle infrastructure for the text file administration of the standard offering.

9.1.10 Use Switched Connections in DMZ

It is recommended that all DMZ attached devices be connected by switched, not bussed connections. Furthermore, devices such as the Cisco 11000 series devices, which can provide IP, port, and protocol rules between each pair of connected devices are preferred.

9.1.11 Place Application Server in the DMZ

Application servers should be in the DMZ. In this architecture Oracle9*i*AS Web Cache only forwards requests to boxes containing Web servers; Web servers only forward requests to application servers (or via PL/SQL to database servers); application servers only forward inward requests to the database or, perhaps, special message processing processors in the intranet. This provides excellent fault containment because, except for PL/SQL which has special security, a compromised Web server must somehow compromise an application server before the database can be attacked - a very difficult and improbable situation.

9.1.12 Tune the SSL SessionCacheTimeout Directive if You Are Using SSL

The Apache server in Oracle9*i*AS caches a client's SSL session information by default. With session caching, only the first connection to the server incurs high latency.

In a simple test to connect and disconnect to an SSL-enabled server, the elapsed time for 5 connections was 11.4 seconds without SSL session caching as opposed to 1.9 seconds when session caching was enabled.

The default `SSLSessionCacheTimeout` is 300 seconds. Note that the duration of a SSL session is unrelated to the use of HTTP persistent connections. You can change the `SSLSessionCacheTimeout` directive in `httpd.conf` file to meet your application needs.

9.2 OC4J Security Best Practices

This section describes OC4J security best practices. The topics include:

- Use the Oracle9iAS JAAS Provider for OC4J User Management in Place of principals.xml
- Avoid Writing Custom User Managers and Instead Extend the JAAS Provider, Oracle9iAS Single Sign-On, and Oracle Internet Directory
- Use Oracle9iAS Single Sign-On as the Authentication Mechanism with the JAAS Provider
- Use the JAAS Provider’s Declarative Features to Reduce Programming
- Use Fine-Grained Access Control Provided by the JAAS Provider and the Java Permission Model
- Use Oracle Internet Directory as the Central Repository for the JAAS Provider in Production Environments
- Take Advantage of the Authorization Features of the JAAS Provider

9.2.1 Use the Oracle9*i*AS JAAS Provider for OC4J User Management in Place of principals.xml

In the earlier releases of Oracle9*i*AS, the J2EE application server component stored all user information in a file called `principals.xml` (including storing passwords in cleartext). The Oracle9*i*AS JAAS Provider provides a similar simple security model as a default without storing passwords in cleartext. However, it also provides tight integration with Oracle9*i*AS Infrastructure (including Oracle9*i*AS Single Sign-On and Oracle Internet Directory) out of the box. Hence, we strongly recommend that you leverage the Oracle9*i*AS JAAS Provider for J2EE security in OC4J.

9.2.2 Avoid Writing Custom User Managers and Instead Extend the JAAS Provider, Oracle9*i*AS Single Sign-On, and Oracle Internet Directory

The OC4J container continues to provide several methods and levels of extending security providers. Although the `UserManager` class can be extended to build a custom user manager, leveraging the rich functionality provided by the JAAS Provider, Oracle9*i*AS Single Sign-On, and Oracle Internet Directory will allow developers more time to focus on actual business logic instead of infrastructure code. Both Oracle9*i*AS Single Sign-On and Oracle Internet Directory provide APIs to integrate with external authentication servers and directories respectively.

9.2.3 Use Oracle9*i*AS Single Sign-On as the Authentication Mechanism with the JAAS Provider

Oracle9*i*AS JAAS Provider allows different authentication options. However, Oracle strongly recommends leveraging the Oracle9*i*AS Single Sign-On server whenever possible for the following reasons:

1. It is the default mechanism for most Oracle9*i*AS components such as Portal, Forms, Reports, Wireless etc.
2. It is easy to setup in a declarative fashion and does not require any custom programming.
3. It provides a seamless way for PKI integration.

9.2.4 Use the JAAS Provider's Declarative Features to Reduce Programming

Since most of the features in the Oracle9*i*AS JAAS Provider are controlled declaratively, particularly in the area of authentication, their setup can be postponed until deployment time. This not only reduces the programming tasks for integrating a JAAS based application, it also enables the deployer to control the same J2EE application via his / her environment-specific security models.

9.2.5 Use Fine-Grained Access Control Provided by the JAAS Provider and the Java Permission Model

Unlike the "coarse-grained" J2EE authorization model as it exists today, the JAAS Provider integrated with OC4J allows any protected resource to be modeled using Java permissions. The Java permission model (and associated Permission class) is extensible and allows a flexible way to define fine-grained access control.

9.2.6 Use Oracle Internet Directory as the Central Repository for the JAAS Provider in Production Environments

Although the JAAS Provider supports a flat-file XML-based repository useful for development and testing environments, it should be configured to use Oracle Internet Directory for production environments. Oracle Internet Directory provides LDAP standard features for modeling administrative metadata and is built on the Oracle database platform inheriting all of the database properties of scalability, reliability, manageability, and performance.

9.2.7 Take Advantage of the Authorization Features of the JAAS Provider

In addition to the authorization functionality defined in the JAAS 1.0 specification, the Oracle9*i*AS JAAS Provider supports:

- hierarchical, role-based access control (RBAC)
- the ability to partition security policy by subscriber (i.e. each user community).

Both of these extensions allow a more scalable and manageable framework for security policies covering a large user population.

9.3 Oracle9*i*AS Single Sign-On Best Practices

This section describes Oracle9*i*AS Single Sign-On best practices. The topics include:

- Oracle9iAS Single Sign-On Servers Should Be Configured for High Availability
- Leverage Oracle9iAS Single Sign-On Whenever Possible
- Have an Enterprise-Wide Directory in Place
- Always Use Oracle9iAS Single Sign-On Instead of Writing Custom Authentication Logic
- For Devloping Single Sign-on Enabled Applications, Use mod_osso and Not the Single Sign-on SDK
- Always Use SSL with Oracle9iAS
- Train Users to be Wary of Providing Their Oracle9iAS Single Sign-On Username and Password Anywhere Other Than Through the Oracle9iAS Single Sign-On URL
- Train Users to Log Out So the Cookie Does Not Remain Active

9.3.1 Oracle9*i*AS Single Sign-On Servers Should Be Configured for High Availability

Single sign-on failure is catastrophic since it means no single sign-on protected application can be accessed. Two recommendations for high availability of Oracle9*i*AS Single Sign-On are:

1. Carefully consider inclusion of any other types of processing on the single sign-on servers since this can make instability more likely.
2. Consider deploying multiple single sign-on servers fronted by load balancing hardware to protect against failures in single sign-on listeners. In this case, the address of the load balancer is used as the single sign-on address and the single sign-on listener configuration information is replicated. It is also recommended that the database be Real Application Cluster (RAC) configured for additional improvements in availability. Configuration details for multiple single sign-on servers can be found in the Oracle Technology Network.

9.3.2 Leverage Oracle9*i*AS Single Sign-On Whenever Possible

Oracle9*i*AS Single Sign-On should be used as the primary point of security. This is a benefit administratively and a major convenience to application customers. Also, Oracle9*i*AS Single Sign-On is well integrated with the rest of Oracle infrastructure and can, via Oracle Internet Directory and other means, be integrated with non-Oracle application and infrastructure. Also, as single sign-on becomes a single point for authentication, opportunities to attack the multiple authentication entities of sites today are reduced. Finally, single sign-on's single authenticated user for all applications allows better control for more uniform authorization.

9.3.3 Have an Enterprise-Wide Directory in Place

In order to deploy an effective single sign-on solution, the user population must be centralized in a directory, preferably an LDAP-based directory such as Oracle Internet Directory. Having users represented in multiple systems (e.g. in multiple Windows NT domains) makes setting up the infrastructure for a common identity more difficult. In addition, clearly defining and automating the user provisioning process makes managing the single sign-on environment much easier.

9.3.4 Always Use Oracle9*i*AS Single Sign-On Instead of Writing Custom Authentication Logic

Oracle9*i*AS Single Sign-On provides the infrastructure to validate credentials and allows for various different authentication mechanisms such as username/password, X.509 certificates. Moreover, since these can be shared across different applications and web sites, end users do not have to create a new username/password for each different corporate application.

9.3.5 For Devloping Single Sign-on Enabled Applications, Use mod_osso and Not the Single Sign-on SDK

Oracle9*i*AS Single Sign-On provides an SDK, which can be leveraged to develop a partner application. This SDK was commonly used in earlier releases of Oracle9*i*AS. However, beginning in Release 2, all of the common usage patterns have been embodied in a new Oracle HTTP Server module, mod_osso, that requires significantly less programming than the SDK. By single sign-on enabling an application with mod_osso, the application will automatically get future enhancements without changing any code.

9.3.6 Always Use SSL with Oracle9*i*AS

The Oracle9*i*AS Single Sign-On server simplifies user interaction by providing a mechanism to have a single username and password that can be used by multiple partner applications. However, with this ease, comes the caution that the single sign-on server should always be accessed in the correct fashion; a breach of the common password can now put all the partner applications at risk. Hence, the single sign-on server should always be configured to allow connections in SSL mode only. This protects the end user's credentials going across the wire. Applications where security and data confidentiality is important should also be protected by SSL. From a performance perspective, use of SSL hardware accelerators is recommended.

9.3.7 Train Users to be Wary of Providing Their Oracle9*i*AS Single Sign-On Username and Password Anywhere Other Than Through the Oracle9*i*AS Single Sign-On URL

The Oracle9*i*AS Single Sign-On server provides a standard login screen. This login page is serviced from the single sign-on server machine, which is typically a different machine from the one the end user is trying to access. Thus, it is critical that before the end user enters her login and password, she ensures that she is at a valid single sign-on screen that looks the same as it always has - from a valid login server computer. This prevents users from unknowingly providing their username/password to inappropriate servers.

9.3.8 Train Users to Log Out So the Cookie Does Not Remain Active

This is generic and not really single sign-on specific, but it is of particular importance when leveraging single sign-on. Most users do not log out of Internet applications and this creates problems at two levels:

1. A security risk - since someone else walking to the station can now reuse the cookie. Also, since the session remains valid until it times out, a hacker from another machine has a longer time window to guess the session id/cookie value.
2. The system resources on the server associated with the cookie are not released until the session is ended or invalidated.

For application developers and administrators, single sign-on session duration and inactivity timeouts should be configured appropriately (for example, one hour inactivity timeouts for sensitive applications).

For external apps, single sign-on cannot logout users from external apps - so closing all browser windows is important in this case.

10

Oracle Enterprise Manager Best Practices

This chapter describes Oracle Enterprise Manager best practices. It features the following topics:

- Use the Oracle Enterprise Manager Job System to Periodically Back Up Your Oracle9iAS Configuration
- Use Oracle Enterprise Manager to Monitor Rate and Aggregated Performance Metrics
- After Restarting Oracle Enterprise Manager, Navigate to Commonly Used Pages
- Use Oracle Enterprise Manager to Change Configurations

10.1 Monitor Application Performance During Application Development or Test Cycles Using Oracle Enterprise Manager

During application development and testing, you can use the Oracle Enterprise Manager Web site to monitor the application's resource usage and identify bottlenecks. For example, during a performance or load test you can view memory and CPU use for the Oracle9*i*AS instance overall and for the application. You can also drill down to find sessions, modules, EJB's, methods, etc., that may be bottlenecks in the application.

10.2 Use Oracle Enterprise Manager to Tune Application SQL

Applications that access the database using SQL can be tuned using Oracle Enterprise Manager SQL tuning tools. During development you can use Oracle SQL Analyze to tune your SQL statements before they are deployed on a test system. SQL Analyze can automatically examine your SQL statements and rewrite the statement to improve performance, such as altering the statement so an index can be used. You can also use Oracle Enterprise Manager to view a graphical display of the execution plan for your SQL statement and compare plans and statistics for different versions of your SQL statement.

During testing of your application you can use Oracle Enterprise Manager to monitor SQL performance and make further tuning improvements. For example, you can use Oracle Expert to recommend a better indexing strategy to improve data access performance.

10.3 Use Oracle9*i*AS Clusters for Application Deployment and Configuration Management

Using Oracle9*i*AS clusters simplifies management and maintenance of your application servers. Clustering enforces consistent configurations across all members of the cluster. So, if you want to make a configuration change in every instance, you only need to make the change once. The clustering mechanism ensures that the new configuration is propagated to all members.

Similarly, clustering also enforces consistency of deployed applications across all application server instances. If you wish to deploy a new application or update an existing deployment on every application server instance in the cluster, you only need to deploy or update the application once. Again, the clustering mechanism ensures that the application is properly deployed to all members. deployment wizard, which can be accessed from the Oracle9*i*AS Instance home page. The wizard walks you systematically through all the essential deployment options to ensure that your application is deployed correctly.

10.4 Use the Oracle Enterprise Manager Deployment Wizard to Deploy Application in Real-Time

A simple way to deploy an application is to use the Oracle Enterprise Manager deployment wizard, which can be accessed from the Oracle9*i*AS Instance home page. The wizard walks you systematically through all the essential deployment options to ensure that your application is deployed correctly.

10.5 Use Oracle Enterprise Manager Job System to Schedule a Deployment to Occur at a Certain Time

In some cases, you may want to deploy an application during off-hours or at a certain scheduled time. You can use the Oracle Enterprise Manager job system to schedule a deployment to occur at a selected time. Simply create a script containing the DCM command-line dcmctl `deployApplication` command and schedule the script via the Oracle Enterprise Manager job system. You can also choose to be alerted when the deployment completes or if there is an error.

10.6 Select the Oracle Enterprise Manager Management Framework Options That Best Suit Your Needs

There are various ways to deploy Oracle Enterprise Manager, which gives you the flexibility to select the configuration that best suits your needs. If you are working in a simple development or test environment, or if you have a single Oracle9*i*AS instance to manage, you would probably need only to install the Oracle Enterprise Manager Web site. The Web site allows you to directly access all the pages for managing and monitoring the instance. This is the simplest management configuration and is automatically installed with all Oracle9*i*AS install types.

In a production environment, you may need to set up events and jobs. To use these capabilities of Oracle Enterprise Manager, you need to install the Oracle Management Server, which is part of the infrastructure installation option. This option also installs the Oracle Enterprise Manager console. The console is a central location from which you can manage your Oracle9*i*AS instances, databases, and your entire Oracle environment. The Oracle Enterprise Manager framework also supports sharing of information between administrators.

10.7 Use the Latest Version of Oracle Enterprise Manager for Managing Both Oracle9*i*AS and the Oracle Database

If you plan to manage both your Oracle9*i*AS instances and your databases from the same management console, install the latest version of Oracle Enterprise Manager. This will ensure that you have the most up-to-date functionality for managing both types of targets.

10.8 Use the Oracle Enterprise Manager Event System and Notification to Proactively Monitor System Availability

The Oracle Enterprise Manager Event System allows you to monitor your systems for specific conditions, such as loss of service or poor performance. You select tests to run on managed targets, such as an application server instance, and then set the threshold parameters for which you want to be notified. Alarms will always display on the Oracle Enterprise Manager console, but you can also be notified via email or page. Minimally, you should set up events to alert you when Oracle Enterprise Manager detects that your critical or production application servers are unavailable.

You can share events with other administrators, in addition to being able to notify specific administrators when an event condition occurs. This simplifies cooperation between administrators who share responsibility for the same systems. For some event tests, you can also choose to execute a fixit job, such as restarting a component that automatically corrects the problem.

10.9 Use the Oracle Enterprise Manager Event Management System's User-Defined Events to Customize Monitoring of Your Application Servers

The user-defined event test allows you to define your own scripts that monitor conditions particular to your environment. These event tests can be written in any scripting language, as long as the node that runs the script has the appropriate runtime requirements to execute the script.

The power and flexibility of user-defined event tests lie in the ability to integrate any custom script into the Oracle Enterprise Manager Event System and leverage the system's multi-administrator, lights-out scheduling and notification capabilities. You can, for example, write a script to monitor the performance of a user application, register that script as a user-defined event, and receive alerts from Oracle Enterprise Manager when performance falls below your specified threshold.

10.10 Use Oracle Enterprise Manager to Monitor and Diagnose Performance Bottlenecks and Availability Problems

Once you have set up Oracle Enterprise Manager to monitor for availability and performance issues, you will be alerted when a problem is detected. If Oracle Enterprise Manager detects that an application server component is unavailable, you can use the Oracle Enterprise Manager Web site to check the status of the component and restart it if desired. If a performance issue was detected, perhaps with a component or application, you can drill down to the component's home page and view detailed performance and diagnostic information. If needed you can also drill down from the OC4J home page to find the most resource intensive applications, modules, methods, etc. Using these drill downs, you can diagnose and resolve performance issues.

10.11 When Monitoring Application Server Performance, Use the Host Home Page to Help Diagnose Performance Issues

The Oracle Enterprise Manager Oracle9*i*AS instance home page not only displays critical performance data and resource usage for the application server instance, it also includes a link to information for the host. For example, if your application server is performing poorly you can first drill down to the related Host home page to determine if the underlying problem is due to resource problems with the host and other processes, or to services running on the box.

10.12 Use the Oracle Enterprise Manager Job System to Periodically Back Up Your Oracle9*i*AS Configuration

Periodically you should back up your application server configuration. By saving your configurations, you can restore the backed up settings if you ever need to undo configuration changes made. You can use the DCM command-line utility's dcmctl saveInstance command in a script to save the configuration and application information for an application server instance. You can then schedule the backup script to run periodically using the Oracle Enterprise Manager job system. This ensures that backups of your configurations are taken on a regular basis.

10.13 Use Oracle Enterprise Manager to Monitor Rate and Aggregated Performance Metrics

Oracle Enterprise Manager's home pages and drill downs include rate and aggregated performance data that are not available via command line or other tools. For example, you can use Oracle Enterprise Manager to view average processing time for a HTTP request, allowing you to zero in on specific requests that may be slow.

Oracle Enterprise Manager also displays performance information, such as average processing time for a servlet for the most recent 5 minutes, in addition to averages since startup. This allows you to more easily diagnose problems in real-time.

10.14 After Restarting Oracle Enterprise Manager, Navigate to Commonly Used Pages

After restarting the Oracle Enterprise Manager Web site, you may want to navigate to commonly used pages on the Oracle Enterprise Manager Web site. This ensures that UI and other software components are pre-loaded. All subsequent accesses to these pages will be faster compared to the first time they are accessed.

10.15 Use Oracle Enterprise Manager to Change Configurations

When you edit the configuration of Oracle9*i*AS components, Oracle HTTP Server, OC4J, or OPMN, you should do so via the Oracle Enterprise Manager Web site. Oracle Enterprise Manager will ensure that your configuration changes are properly updated in the repository. If you edit these configuration files manually, you must use the DCM command-line utility (dcmctl) to notify the DCM repository of the changes.

11 Installation Best Practices

This chapter describes installation best practices. It features the following sections:

- General Installation Best Practices
- Hosting Installation Best Practices

11.1 General Installation Best Practices

This section describes general installation best practices. It features the following topics:

- Understand the Various Configuration Tools Available with Oracle9iAS
- Try Standard Demos and Associated Applications Before Running Your Applications
- Turn Off Unused Services to Reduce Oracle9iAS Memory Requirement
- Check Metalink Regularly for Updates to Keep Your Installation Current
- Periodically Check the Log Files for Restarts / Errors That Are Masked by Auto Restart Capability
- System Administrator and Oracle9iAS Administrator Should Be Different
- Use the Appropriate Administration User Accounts
- Install All Mid-Tiers on Multiple Smaller Machines, the Infrastructure on Clustered Larger Machines
- For a 3-Tier Environment, Install the Infrastructure Instance Twice and Configure Each Tier Differently
- Recommendation for Installing Oracle9iAS Portal

11.1.1 Understand the Various Configuration Tools Available with Oracle9*i*AS

The appendices of your *Oracle9i Application Server Installation Guide* Release2 (9.0.3) provide detailed documentation of the Oracle9*i*AS Configuration Assistants. Different installations will use different configuration assistants depending on configuration options. For more information, also look at the “Troubleshooting” appendix in these guides.

11.1.2 Try Standard Demos and Associated Applications Before Running Your Applications

This helps segregate the installation time problems from the application problems clearly. Additionally, these demos may be a good way to verify your setup on an ongoing basis before debugging application specific issues.

11.1.3 Turn Off Unused Services to Reduce Oracle9*i*AS Memory Requirement

Oracle9*i*AS supports a wide variety of applications and services. However, your particular deployment or use may not need all of them. Hence, it is a good idea to turn off the unused services. This reduces the memory requirement of Oracle9*i*AS and is a recommended security best practice.

11.1.4 Check Metalink Regularly for Updates to Keep Your Installation Current

Oracle Metalink contains a wealth of support related information and also includes the patches that may be released from time to time. These updates will ensure that you do not run into the same issues that Oracle may have already resolved. While there is benefit to not perturbing a working system and thus, some inertia in updating with the latest patches, at the least, security patches should be applied immediately. Another good source of information is the installation FAQ on Oracle Technology Network.

11.1.5 Periodically Check the Log Files for Restarts/Errors That Are Masked by Auto Restart Capability

With Release 2, Oracle9*i*AS includes improved fault monitoring and recovery capabilities. The monitoring processes automatically restart a failed component. This capability is extremely useful for high availability. However, it could mask the application failures that should be looked at and fixed. Hence, it is an important practice to periodically scan for these auto-restart logs within the log file.

11.1.6 System Administrator and Oracle9*i*AS Administrator Should Be Different

In most deployment scenarios, the operating system and hardware are managed by different teams. Since almost all tasks associated with Oracle9*i*AS installation and configuration can be performed by a user account without acquiring a systems 'root' privileges, it is required to have a separate account to install and configure Oracle9*i*AS.

Moreover, as you look ahead to install multiple instances of Oracle9*i*AS on the same machine, it is prudent to have a separate user for each. This further makes it easier to segregate errors and delegate management for each instance to a different user ID. Ensure that the second or subsequent install users have privileges to write to `targets.xml` file in the first installation. (Note: The Oracle Enterprise Manager user is per node. Thus, having different system users (for example, on UNIX) does not help the cause of having different users manage the different instances over the web. This feature is targeted for a future release.)

11.1.7 Use the Appropriate Administration User Accounts

As the number of installations in an environment grows, it is important to either use similar user names on different machines or, have an algorithm to determine the same. Moreover, each instance has a number of user accounts required by Oracle9iAS itself, such as `orcladmin`, `ias_admin`, portal admin, sso admin, web cache admin, etc. (refer to the *Oracle9i Application Server Administrator's Guide* for details of these accounts). Hence, it is a good idea to have an internal matrix covering these different user types.

11.1.8 Install All Mid-Tiers on Multiple Smaller Machines, the Infrastructure on Clustered Larger Machines

Oracle9iAS mid-tier supports clustering smaller machines into a larger unit. While these machines can coordinate process state and routing information with each other, an individual machine is self-sufficient in processing the requests routed to it. It has no physical dependency on other mid-tier machines. Thus, several of these machines can effectively be used to satisfy a larger number of requests providing a better alternative to more expensive hardware clustering.

On the other hand, the infrastructure installation contains the repository - physical data store that has to be shared across several mid-tier instances. This metadata service needs to present the same data to all mid-tier instances and needs to be up all the time. Hence, it is recommended that the infrastructure installation be carried out on a hardware cluster (for example, Sun cluster, HP Service Guard).

11.1.9 For a 3-Tier Environment, Install the Infrastructure Instance Twice and Configure Each Tier Differently

Most security policies will prevent the Oracle9*i*AS metadata service and repository from being implemented in the DMZ in a 3-tier installation: Web server, J2EE tier, and data store. However, Oracle9*i*AS Infrastructure includes a security service for single-sign on that has to be accessible from the external world.

Hence, we recommend that you install Oracle9*i*AS Infrastructure both behind the firewall and in the DMZ. Only the security service will need to be configured in the DMZ layer, while only the metadata repository Oracle Internet Directory needs to be configured in the layer behind the DMZ. This setup requires some manual steps since, by default, the management console assumes both are deployed on the same Infrastructure instance.

11.1.10 Recommendation for Installing Oracle9*i*AS Portal

When installing Oracle9*i*AS Portal through the Portal and Wireless or Business Intelligence and Forms install types, Oracle9*i*AS Web Cache is installed and configured. If you do not use an Oracle9*i*AS Infrastructure for your Oracle9*i*AS Portal metadata and use another existing database, then de-select Oracle9*i*AS Portal during the installation process. After installation, launch the Oracle9*i*AS Portal Configuration Assistant. Instructions for doing this are in the *Oracle9iAS Portal Configuration Guide*.

11.2 Hosting Installation Best Practices

This section describes hosting installation best practices. It includes the following topics:

- Install as Different Users When Installing Multiple Instances on the Same Machine
- Share the Same Security Service Across Multiple Installations But Split the Metadata Service
- Recommendations for Having Large Number of Groups Run the Applications on a Given Instance

11.2.1 Install as Different Users When Installing Multiple Instances on the Same Machine

While this is a relevant installation practice, it is particularly important in a hosting environment. Doing so lets you provide the hosting customers (who “own” this installation) to access their log files or deploy their own applications. You can always prevent the configuration files from being edited by making them read-only.

11.2.2 Share the Same Security Service Across Multiple Installations But Split the Metadata Service

In most hosting environments, enterprises provide similar capabilities of certain applications to both their intranet and Internet users. This requires some consolidation of services that can be shared to reduce the workload. Security service is one such example. The different deployments can all point to the common single sign-on security service, thus avoiding the need to maintain distinct or similar user IDs in the different security services. Hence, an intranet user may log into the Internet system by using the same username/password although this user’s privileges for the application is governed by the Internet application metadata.

11.2.3 Recommendations for Having Large Number of Groups Run the Applications on a Given Instance

Most large enterprises have a separate organization providing the application server runtime environment as a service to individual organizations in the enterprise. The latter in turn own the applications running on top of the environment.

In this scenario, multiple application server instances can be installed and provided to each target group. However, this increases the maintenance overhead and should be avoided unless security or resources sharing is of paramount concern (as is the case for an external ASP). The concerns here are:

- One organization’s bad application should not take down the servers hosting other divisions’ applications.
- Each division should be able to control its applications.

Oracle recommends that you have several independent OC4J instances within an Oracle9*i*AS instance and dedicate each instance to an organization. Depending on the organization's needs, the number of processes within the OC4J instance can be varied. Since the J2EE configuration is OC4J instance specific, this gives a fair amount of leeway to each organization for their applications.

12 Deployment Best Practices

This chapter describes deployment best practices. The topics include:

- Deployment Architecture Options
- General Deployment Best Practices
- Oracle Internet Directory Deployment Best Practices

12.1 Deployment Architecture Options

The Oracle9*i*AS J2EE and Web Cache install type provides several choices for deployment. These are different and there are trade-offs. Thus, there is no single recommended way to deploy the components.

This section covers a few deployment architectures. It is not possible to cover all possible scenarios, but the discussion associated with each architecture will allow you to create a deployment architecture best suited to your environment.

This section contains the following topics:

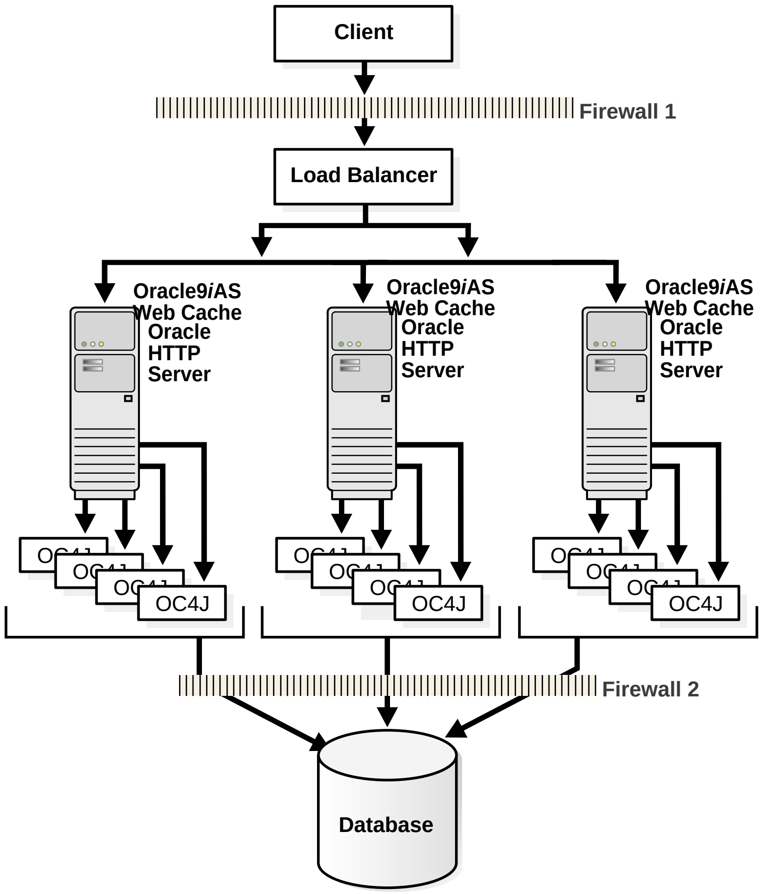
- Deploying Oracle9iAS as Independent Instances
- Deploying Oracle9iAS Instances with Oracle9iAS Web Cache Cluster
- Use Standard Oracle9iAS Clusters
- Separate OC4J and Oracle HTTP Server Clusters

12.1.1 Deploying Oracle9*i*AS as Independent Instances

This is the most common deployment scenario and has the least setup overhead.

It is also the only available option for Oracle9*i*AS instances based on install types other than J2EE.

Figure 12–1 Independent Oracle9iAS Instances Sharing an Infrastructure



In this scenario, independent Oracle9*i*AS instances are installed on independent (identical) machines. These installations share the same infrastructure database. These multiple installs are glued together with a (fault tolerant) load balancer in the front, which routes the request to the different back-end servers on a round robin basis. Sticky session based routing will be required on the load balancer.

The biggest benefit of this approach is that everything works out-of-the-box, and the only task is application deployment on the back-end machines. The load balancer provides balancing across machines, the Oracle9*i*AS Web Cache on each Oracle9*i*AS instance provides the basic caching for each instance, and the Oracle HTTP Server load balancing component (mod_oc4j) provides load balancing across the individual OC4J processes on each machine.

There are few drawbacks, such as:

- The Web caches are not clustered, and do not benefit from the data cached by other caches.
- OC4J processes between machines do not replicate state. Thus, the HTTP conversational state is lost on a machine failure.
- A J2EE application has to be deployed (or other configuration changes have to be made) on each instance – as opposed to doing it once and propagating it across the cluster.

However, this option is attractive due to its simplicity. And if you practice state-safe programming as recommended in this document, some of the drawbacks become even less relevant.

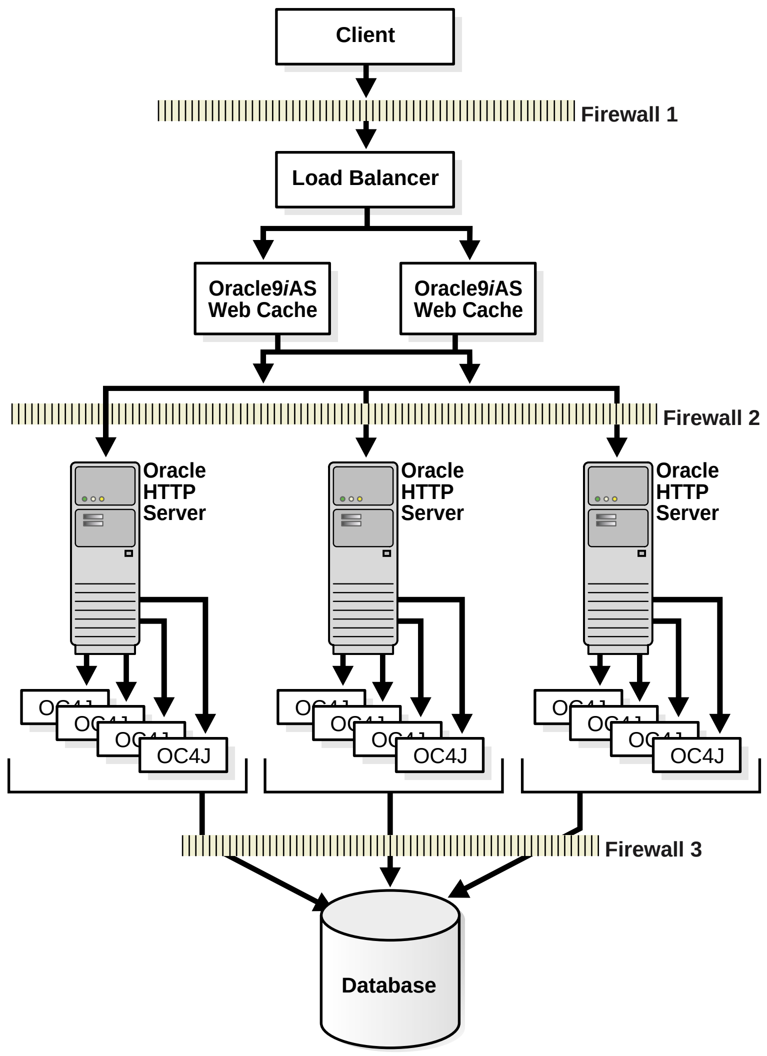
You can however do some non-standard manual tweaks to remove this restriction, as documented in the mod_oc4j Technical Overview paper at:

`http://technet.oracle.com/products/ias/ohs/collateral/r2/mod_oc4j_wp.pdf.`

12.1.2 Deploying Oracle9*i*AS Instances with Oracle9*i*AS Web Cache Cluster

Let us say you have the earlier deployment architecture working right, and now you want to gain the benefits of Oracle9*i*AS Web Cache clustering and/or introduce an additional security zone.

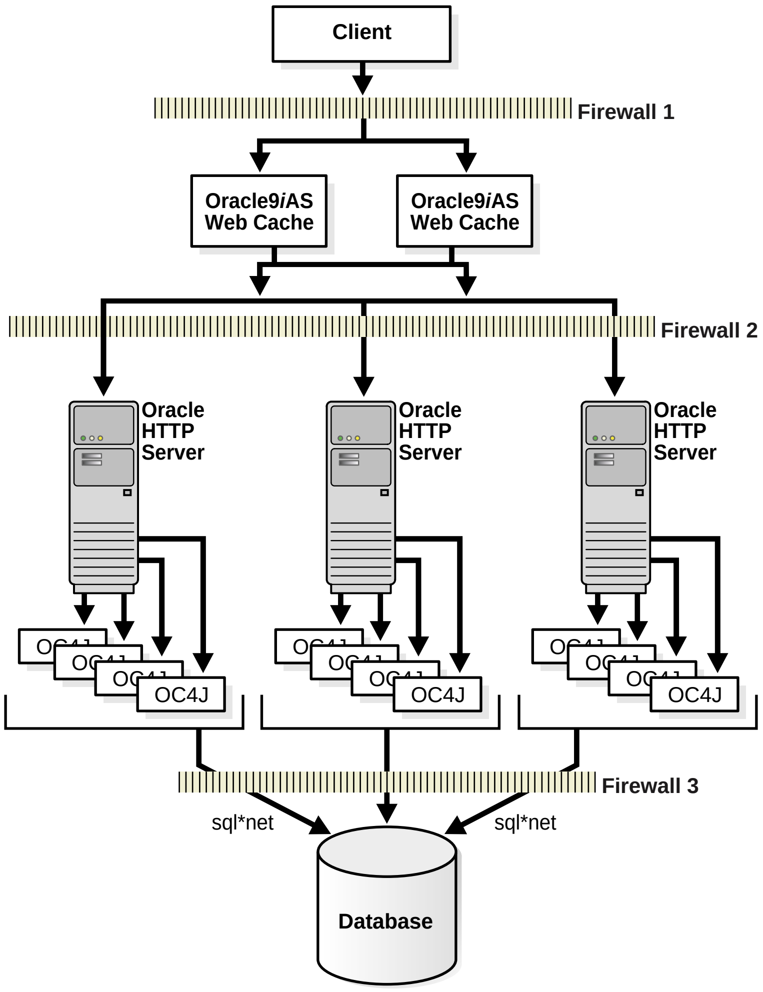
Figure 12–2 Separate Oracle9iAS Web Cache Instances



The architecture shown in Figure 12–2 allows that. You just need to shutdown the Oracle9*i*AS Web Cache service on the individual instances on the back-end machines, and move those instances onto separate suitable machines. These could then be clustered, so that the different Oracle9*i*AS Web Cache instances can provide added fault tolerance and scalability, along with the ease of configuration.

The load balancer and the Oracle9*i*AS Web Cache instances have been moved into a security zone of their own: providing added security check before the backend systems (Oracle HTTP Server and OC4J) can be reached. This architecture is shown in Figure 12–3.

Figure 12–3 Separate Instances Without Load Balancer

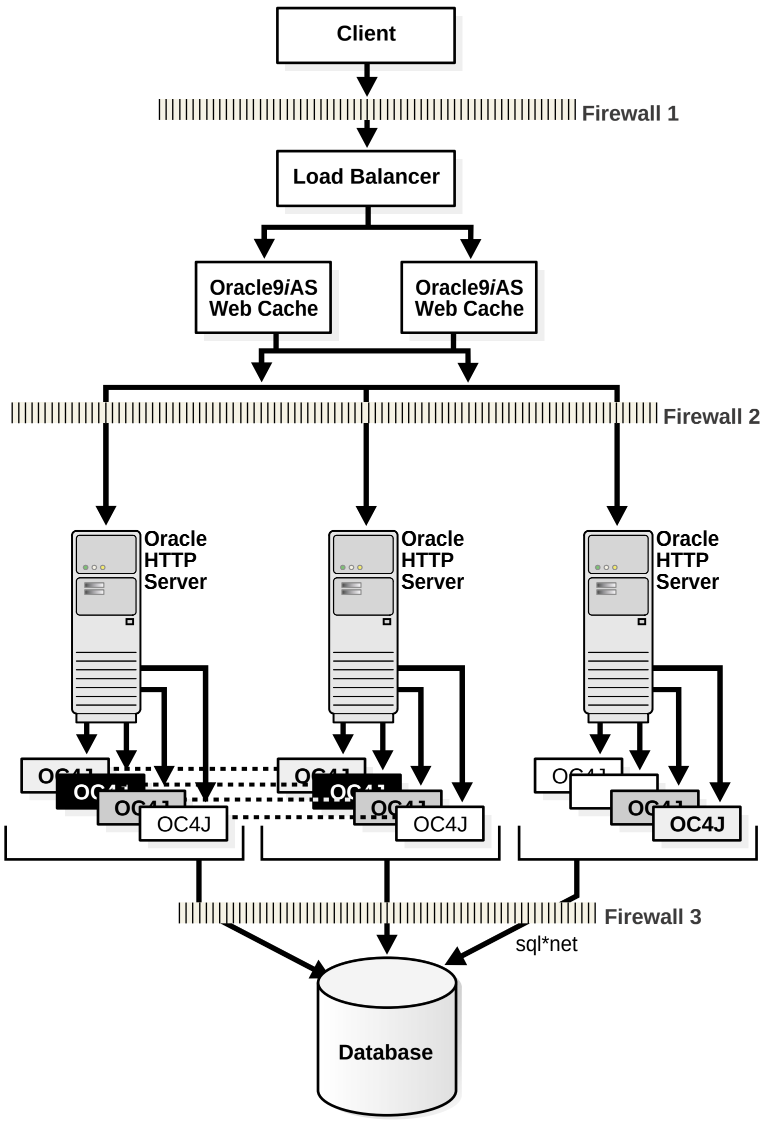


Another variation, as shown above is the removal of load balancer, and using a single Oracle9*i*AS Web Cache instance to balance across the multiple back-end origin servers. This is a cost-effective option since it does not require an expensive load balancer in front. However, although it violates one of the best practices discussed in the Oracle9*i*AS Web Cache chapter.

12.1.3 Use Standard Oracle9*i*AS Clusters

Oracle9*i*AS Release 2 introduces powerful clustering technology, that allows for session replication across OC4J instances, dynamic restarts, cluster-wide configuration updates, etc.

Figure 12–4 OC4J and Oracle HTTP Server Clusters



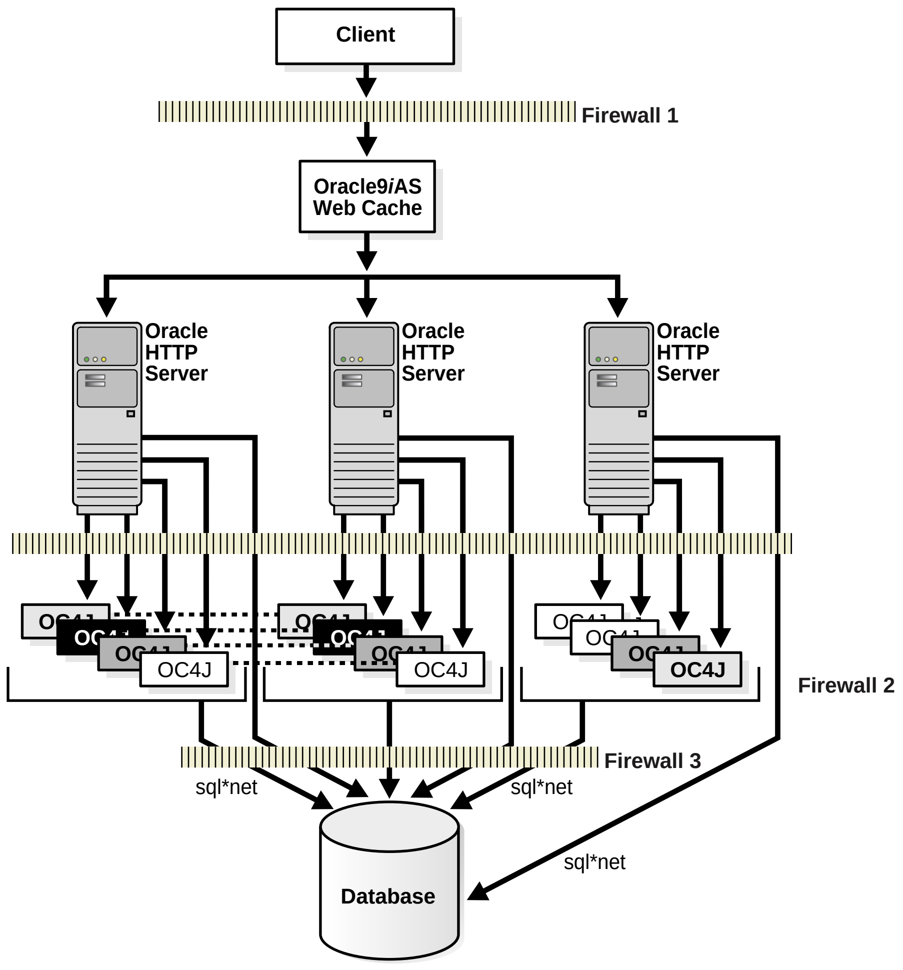
In Figure 12–4 above, the previous architecture was modified such that all the Oracle HTTP Server and OC4J are clustered together. Within this cluster, you can select groups of OC4J processes (instances) to be part of the same island so that they can provide failover for each other.

It is recommended that this island size be 2 to 3 which is much smaller than the overall cluster size. It reduces the session replication overhead. This scenario provides better fault tolerance than any of the earlier scenarios and is the best option of all recommendations, especially for J2EE and Web Cache deployments.

12.1.4 Separate OC4J and Oracle HTTP Server Clusters

Sometimes it is desirable to separate Oracle HTTP Server from OC4J (the application server component) into different security zones. This scenario is shown in the following Figure 12–5.

Figure 12–5 OC4J and Oracle HTTP Server Separate Clusters



It contains two clusters: a cluster containing only Oracle HTTP Server instances and a cluster containing only OC4J components.

If desired, Oracle9*i*AS Web Cache can also be clustered, but that will necessitate a load balancer in front of the Web cache servers.

This scenario requires a fair amount of hand modifying configuration files, so that the two clusters (Oracle HTTP Server and OC4J) can communicate with each other. Moreover, it also requires opening of sql*net and LDAP ports on FW-2, in addition to FW-3.

This scenario also does not benefit from the distributed deployment due to the required manual edits.

These drawbacks make this option less appealing – simply to move Oracle HTTP Server out into its own security zone. Hence, we recommend the architecture shown in Figure 12–4 as the preferred method.

12.2 General Deployment Best Practices

This section describes general deployment best practices. It contains the following topics:

- Deploy / Re-Deploy Applications During Low Traffic
- Use Identical Machines
- Rolling Upgrades - Form a New Cluster for Major Upgrades
- Use Network Level Load Balancer for Increase Scalability / Availability

12.2.1 Deploy/Re-Deploy Applications During Low Traffic

Oracle9*i*AS supports applications to be deployed on a production system without taking the system down.

However, some applications (e.g. EJB-based EAR files) may put additional load on the production system by requiring compilations, JAR file updates, and Web server restarts. While this doesn't result in any lost requests directly, it does introduce some additional load on the system.

Hence, it is a good practice to avoid deploying during high traffic periods.

12.2.2 Use Identical Machines

Oracle9*i*AS clustering allows the ability to mix-and-match different hardware in the same cluster. While this may be attractive and allow for some innovative solutions, it makes overall manageability and operations harder. Hence, it should be avoided.

Having identical hardware allows for ease of stocking spare parts, better reproducibility of problems (both software and hardware), and reduces the per-platform testing cost. This cost, in many cases, will surpass the savings from re-using existing disparate hardware.

12.2.3 Rolling Upgrades - Form a New Cluster for Major Upgrades

Oracle9iAS with its distributed configuration management capability makes it easy for several machines in the cluster to have the same configuration.

However, there is a performance and possible downtime cost associated with deploying applications to a cluster. To reduce the impact of this, starting a new cluster is recommended in the following scenarios:

- If your cluster size is big.
- If you are going to be deploying quite a few applications.
- If you are making quite a few configuration changes.

Specifically, the recommendation is:

1. Remove one node from the existing cluster.
2. Make the changes (deploy applications, configuration changes, etc.) on this node.
3. Start a new cluster Oracle9*i*AS Web Cache– say new-Production – with just this node as the member.
4. Now remove nodes from the Production cluster one by one and add them to the new-Production cluster. Removing a node from a cluster does not cause any significant performance impact. The node will see performance degradation (multiple deployments, configuration changes etc.) only when it is joining the new cluster. However, since it is not receiving requests the overload will not impact production.
5. Once the new-Production reaches sufficient membership, swap the two clusters (by either changing the Web cache or external load balancer configuration) to point to the new-Production machines.

This practice allows you to make major changes to the system without impacting all machines in the cluster at once. It also results in changes being applied more reliably.

12.2.4 Use Network Level Load Balancer for Increase Scalability/Availability

Load balancers like Cisco’s CSS 11000 and F5’s BigIP can quickly detect and route around failed mid-tier servers. For greatest failure isolation, each server should be an independent computer, although if the availability of the computer and its operating system is very high, multiple servers may run on the same computer. Load balancers can also quiescence and take a mid-tier server off-line, so that it can be restarted or reconfigured without having to handle incoming requests at the same time.

Ideally, N+2 mid-tier machines should be used, where N is the number of mid-tier computers needed to serve peak load. One extra computer is a ready-to-go spare, should one of the N fail. The other extra computer hosts a server that is off-line for restart or maintenance.

Load balancers have failover configurations to prevent themselves from becoming single points of failure.

Load balancers can also be used to balance traffic across multiple firewall computers for scalability and availability. This can be a complex and expensive option, because you must have a load balancer in each security zone. Most commercial firewalls have a failover solution that does not require use of a load balancer, so unless scalability is a concern, use the availability solution recommended by the firewall vendor.

12.3 Oracle Internet Directory Deployment Best Practices

This section describes the Oracle Internet Directory deployment best practices. It includes the following topics:

12.3.1 Use Utility for Bulk Loading Data to Oracle Internet Directory

There is a bulkload utility (`$`*`ORACLE_HOME`*`/ldap/bin/bulkload.sh`) that can be used for loading large amounts of data, for example, from existing third party directories. The utility also has a schema check option, to avoid difficult to trace errors. For smaller amounts of data (less than 10,000 entries), use the `ldapadd` utility as it is adequate for the job. If bulkload utility is not used, you should execute `$`*`ORACLE_HOME`*`/ldap/admin/oidstats.sh` to collect database statistics so that the LDAP search operations perform optimally.

The bulkload utility takes a `ldif` file as input data file. When generating these files from third party directories, you will have to scrub out some of the operational attributes as these are generated by Oracle Internet Directory during bulkload. However, if the input `ldif` file is from another Oracle Internet Directory instance, then you must use the restore option in `bulkload.sh` to preserve these operational attributes as-is during bulkload.

12.3.2 Replicate Oracle Internet Directory for High Availability

To hide multiple Oracle Internet Directory nodes from client applications, a BIG-IP or similar device can be used to load balance or provide a single IP and port to the client applications. If a replicated node goes down, the client applications do not see the failure, because the BIG-IP device re-routes requests transparently.

Additionally, each Oracle Internet Directory node can run on Oracle9*i* Real Application Clusters (RAC). This further improves availability regardless of whether a BIG-IP set up is used or not. The RAC instances provide increased database and data availability while presenting the same data via a shared disk mechanism.

Other high availability solutions deployable with Oracle Internet Directory are hardware cluster failover and Oracle Data Guard.

12.3.3 Use SSL Binding for Better Security

SSL is considered de facto Internet standard protocol for highly secure transportation of data. In addition to the strong PKI authentication using digital certificates, SSL also provides multiple data integrity and data encryption layers to protect your communication channels. SSL provides multiple cipher suites with varieties of encryption algorithms for many security levels.

Oracle Internet Directory supports 3 SSL authentication modes:

1. Confidentiality mode (no-authentication mode)

   In this mode, SSL cipher suites use the Diffie-Hellman algorithm to generate a session key for client or server at run time. The session key will be used to encrypt the communication channel. No server or user SSL wallet is necessary. In SSL no authentication mode, the channel will be encrypted using a Diffie-Hellman key.

2. Server Authentication only mode

   This mode essentially uses certificates for authentication. The client needs to verify the server certificate. This mode is most commonly used in the Internet environment since any client that needs to talk to a SSL server does not require a certificate. A client can use its user / password to authenticate itself to the server. The username and password are protected by SSL encryption when being transferred on the wire.

3. Server and Client Authentication mode (Mutual authentication)

   In this mode, both client and server use RSA certificates to authenticate each other. First, the client authenticates the server by validating its certificate. In return, the server also requires the client to send its certificate to prove its authenticity.

In addition to choosing an authentication mode, one should choose appropriate security algorithms. Following are the strong security algorithms supported in Oracle Internet Directory:

- Data integrity algorithm: SHA1
- Data privacy (encryption) algorithms: 3DES, and RC4 –128 bit encryption algorithms

12.3.4 Use the Appropriate Backup and Restore Utilities

Oracle Internet Directory supports two backup and restore mechanisms for this: database export import based mechanism, and LdifWriter and bulkloader based mechanisms.

The former is faster, but it is not flexible - you have to backup / restore the entire directory, and this cannot be done incrementally. The latter utilities, LdifWriter and bulkloader provide more control to restore / backup subsets of the directory. However, they are slower and require more manual steps.

Hence the recommendation is to use the database-based utility when doing a complete backup-restore, and use the other utilities when you intend to backup and restore a subset of the directory.

No matter which mechanism is used, make sure the backup-restore is used for the same versions. For migration purposes - to a different version of Oracle Internet Directory - ensure you follow the appropriate instructions before bulkloading or restoring the directory data.

12.3.5 Recommendations for Audting and Monitoring Oracle Internet Directory

You can monitor and audit Oracle Internet Directory in one of three ways:

1. EM's LDAP page provides a very simple way to monitor the LDAP service and determine if it is up and running and its associated load.
2. You can also check the log files of various LDAP processes to ensure there are no spurious errors showing up.
3. LDAP audit log service provides more granular information such as security violation information or sensitive events. The audit log can be further customized to specific directory operations and events.

Oracle recommends that you perform, at the very least, a weekly review of the audit and error logs. System administrators can do a more regular review via the EM health check to provide better availability.

12.3.6 Assign Correct Oracle Internet Directory Privileges for Oracle9*i*AS Installation

While it is possible to install Oracle9*i*AS as an Oracle Internet Directory super user, we highly recommend that this not be done as it provides the user installing Oracle9*i*AS more privileges than required.

To install Oracle9*i*AS, a user needs to be a member of Oracle9*i*AS Administrator’s group and should also be an owner of that group.

While installing Oracle9*i*AS, the directory administrator should add the installation user as a member and owner of the Administrator’s group, and then remove him/her as the owner once the installation has completed so that the user does not have privileges to perform any more installations.

12.3.7 Change Access Control Policies to Better Administer Users

By default, Oracle Internet Directory is installed with the following privileges for the Oracle Internet Directory administrator: create new users, delete existing users, and modify user attributes.

The access control setup on a user’s container specifies the default access control policies regarding user administration. Oracle Internet Directory administrators should change the default access control policies to better control user administration as per required.

12.3.8 Best Practice for Oracle Internet DirectoryOracle Internet Directory Password Policy

Oracle Internet Directory supports both password value policies and state policies. The password policies can be enforced on a per-subscriber basis in a hosted environment. If a given subscriber does not have any applicable policy, the policy under the root Oracle context will be applied to that subscriber.

The value policies are: password minimum length, minimum number of numeric characters, disallow the use of current password as new password, and disallow the use of common words and attribute values as passwords.

The state policies supported are: password expiration, account lockout, expiration warning, and grace logins.

The best practice password policy is to have a password minimum length of 5 characters with at least one numeric character. Also, it is recommended to have a password expiration duration of 60 days and lock user accounts after 10 consecutive login failures. The user account must stay locked for a duration of 24 hours. Additionally, users must be sent password expiration warnings at least 3 days prior to expiration and should be allowed a maximum of only three grace logins after their passwords have expired.

12.3.9 Best Practice for Directory Integration Platform in Third Party Directory Environments

This section features the following topics:

- Identity Provisioning
- Synchronization Configuration
- Oracle HR Synchronization

12.3.9.1 Identity Provisioning

Directory Integration Platform (DIP) should be used to build connectivity between Oracle Internet Directory and third party directories so that other Oracle products can seamlessly work in presence of third party directories in the enterprise and can also share the same identities with other directories.

The different identities for the same enterprise user from multiple LDAP directories can be joined / unified into a single global identity in Oracle Internet Directory using DIP, which would facilitate a true single sign-on environment in an enterprise using Oracle Internet Directory / Oracle9*i*AS Single Sign-On.

Oracle Internet Directory supports representation of multiple applications and multiple subscribers (administration Contexts) in the Oracle Internet Directory DIT. Various enterprise applications can be provisioned for a single or multiple subscriber(s). There are automated tools to create new Subscribers (administrative Contexts) and to provision applications for various subscribers. These tools setup the various levels of access required by the application to manage the Subscriber.

Data from third party directories should be synchronized with Oracle Internet Directory into the appropriate subscriber (administrative context) area so that identity management and group management required by all enterprise applications can be done without any extra overhead.

12.3.9.2 Synchronization Configuration

The scheduling interval of the profiles, which decides the frequency of synchronization, should be set to a value based on the enterprise policy in supporting new users in different environments.

If only selective containers within a domain need to be synchronized, then it will be better to specify the selective container names in the domain mapping rules rather than specifying that top level domain name.

The number of attributes to be synchronized needs to be decided based on the purpose of synchronization. It’s better to specify the minimal set of attributes to have a minimal network and system loads. If the main purpose is to enable the users for using applications such as Oracle9*i*AS Portal, a minimal set of attributes containing the essential user information to be displayed in DAS should be configured for synchronization.

A synchronization Profile has to be DISABLED before altering any STATUS attributes through the ODM. After the change, it needs to be ENABLED once again.

12.3.9.3 Oracle HR Synchronization

Since the “Last Successful Execution Time” is used to fetch changes from the Oracle HR Database, to get the entire user population, this attribute should be set to some old date. Then the profile needs to be ENABLED.

It is a good idea to synchronize user data from the HR database to the public users container in the subscriber area in the Oracle Internet Directory DIT. This way, all the users are immediately accessible to the Oracle9*i*AS Single Sign-On and DAS.

The User Nickname Attribute should be synchronized from the Oracle HR Database or derived from some other attribute (by mapping) which is unique in the HR database in order to uniquely identity a user to Oracle9*i*AS Single Sign-On.

Since the HR Synchronization Profile needs privilege to read write the public users area under the subscriber, the HR profile DN should be added to the `DASCreateUserGroup`, `DASEditUserGroup`, `DASDeleteUserGroup` groups for that subscriber.

12.3.10 Recommendations for Migrating Oracle9*i*AS Application to Existing Oracle Internet Directory

Oracle9*i*AS provides an Oracle Internet Directory instance as part of its infrastructure installation. However, some organizations may already have an existing Oracle Internet Directory installation that houses corporate data or they may have been running the previous versions of Oracle9*i*AS that already have some of the information duplicated. In this case, if it is desired that Oracle9*i*AS should point to the different Oracle Internet Directory, the following should be adhered to:

1. Before starting the upgrade process, the user keys in the older repository must be made consistent with the keys used to identify users in Oracle Internet Directory. This will enable the upgrade process to correlate the private keys with those present in the production Oracle Internet Directory system.
2. The upgrade process will install a private infrastructure (including a private Oracle Internet Directory) against which the component can be validated before switching to the production version of Oracle Internet Directory in the deployment.
3. Once the upgrade is completed, the upgraded component should be verified for correctness.
4. Once the verification is complete, migrate applications from using the private Oracle Internet Directory to the production Oracle Internet Directory.

In case the corporate directory is replicated, some special steps need to be taken by administrators to create a test replica of the production Oracle Internet Directory. Then, migrate and verify the components against the test replica before switching to use the production Oracle Internet Directory.

12.3.11 Recommendation for Moving Oracle9*i*AS Applications From Test to Production Oracle Internet Directory

Like the upgrade case, the presence of a corporate directory in a deployment influences the process by which the deployment can roll out new services using Oracle9iAS. The following steps should be considered when deploying an Oracle9iAS component that uses Oracle Internet Directory:

1. The install process will install a private infrastructure against which the component can be validated before switching to the production version of Oracle Internet Directory in the deployment.
2. Once the install is completed, the component should be verified for correctness.
3. Once the verification is complete, migrate the application from using the private Oracle Internet Directory to the production Oracle Internet Directory. In case the corporate directory is replicated, some special steps need to be taken by administrators to create a test replica of the production Oracle Internet Directory. Then, install and verify the components against the test replica before switching to the production Oracle Internet Directory.

13

Miscellaneous Best Practices

This chapter describes miscellaneous best practices. It features the following sections:

13.1 Simulate Failures and Compute Availability Impact

For this exercise, adopt a pessimistic, Murphy's Law attitude: if it can break, it will! Power off server machines, routers, load balancers, and firewalls. Unplug network cables. Unplug disk drives.

For each failure, does fail over occur automatically? How long does it take a system administrator to locate the failure? Ideally, a management framework will issue specific alerts targeting the failed component until it is repaired. After a real failure, be sure to order replacement parts promptly.

Finally, what is the impact of repair? Are components hot-pluggable so that the repair can be effected without shutting down other components? Repair, reconfiguration, or just adding a new server to a cluster to increase capacity can be major sources of downtime. Note that if components are not hot-pluggable, you should be careful not to cause real failures when unplugging components for this exercise.

Having experienced a simulated failure in training can greatly decrease the risk of a bad problem being made worse by inexperienced operations staff under pressure.

13.2 Pooling and Sharing

Constant creation and destruction of resource objects can be very expensive in Java. Having a resource pool for such resources to be shared across clients can have significant performance & scalability gains:

- Shared resources are used by multiple clients at the same time (parallel re-use), for example: read-only query results from the database
- Pooled resources are used by one client at a time (serial re-use), for example: JDBC connection pools, stateless Session Beans, and servlets/JSPs.

Following are some objects that can either be cached / shared / pooled:

- BP-JDBC connections and statements can be pooled and used across client requests.
- Results of expensive queries can be pooled. Read-only queries can even be shared concurrently too.
- Computed Java / XML objects can either be pooled or shared depending upon their statefulness.
- Parsed XML files and XSL stylesheets.
- Output of HTTP requests in Oracle9*i*AS Web Cache, Akamai cache, Inktomi cache, etc.

You should analyze your system to determine other domain objects that may meet these criteria.

13.3 Perform Incremental Performance Evaluation During the Development Cycle

Do not wait until the end of the project to do a test cycle. Performance tests should be run regularly after each stage of development. New performance test suites should be added as new features or functions are implemented. If possible, run performance regression tests regularly to compare performance results.

13.4 Run Your Performance Test on Systems That Will Simulate Your Production Environment

Developers commonly run their functional tests in single-user mode on their workstation or their desktop, which usually has only one CPU. This setup rarely represents the production environment, and it is not adequate for running performance tests. If the application is intended to be used by a large number of concurrent users running on multiple processors, simulate the production environment with a representative workload to study the performance impact.

13.5 Understand How to Configure Your Test Driver and Analyze the Result

Commercial drivers used to simulate HTTP requests can be very effective, but they are often complicated to configure. Oracle has encountered situations where customers set up their driver incorrectly, did not know how to interpret the results, ended up drawing the wrong conclusions, and wasted valuable time in locating the real problems.

13.6 Assign Someone Who is Experienced in Running and Analyzing Performance Tests

Running performance tests is not a push-button job that can be delegated to an inexperienced engineer. It requires someone who has knowledge and experience with operating systems and databases, and who understands the application that is to be analyzed.

13.7 Document All Recovery and Repair Procedures, and Practice Them Regularly

During a failure, operations staff will be under pressure. It may be difficult to think clearly. It is not a good time to try a new, risky, or unfamiliar repair procedure.

Document the following:

- Backup files and archived log locations
- Diagnostic tool syntax
- Location of spare parts (disks, network cards)
- How to replace failed parts (what should be powered off/on, what racks, chassis, or slots. hold what components
- what needs to be restarted
- contacts: supervisors, support personnel, other experts

Conduct periodic fire drills so when a real failure occurs, it will not be the first time the staff has had to react under pressure. Try to simplify complex repair procedures to minimize errors.

13.8 Use Available Tools to Monitor Site Load and Status

Oracle9*i*AS includes tools that provide good status updates without impacting the production instance.

Some of these (for example dynamic monitoring service) are well integrated with Oracle Enterprise Manager and provide graphic updates on the Oracle Enterprise Manager console. Others (for example iHAT, Oracle9*i*AS Cluster Monitor tool) are available as utilities on the Oracle Technology Network Web site.

Collectively, these tools provide with a means to monitor a production instance while not requiring you to be physically on the machine. It is a good practice to use these, in addition to other system level utilities or tools you may already have in place.

13.9 Rolling Period Restarts Avoid Unexpected Errors

It is generally a good practice to bounce servers periodically. This recovers slow memory leaks, temporary disk space build-up, or other hidden problems that may manifest themselves only after long durations. This is a simple way to avoid unexpected failures.

Oracle9*i*AS makes it easy to follow this practice without requiring a client visible system downtime. You should setup a cluster of servers and setup a staggered reboot schedule for the individual servers.

When a process is down, Oracle9*i*AS automatically takes it out of its routing structure. For http requests, this is ensured by Oracle9*i*AS Web Cache and for J2EE requests this is ensured by mod_oc4j and the clustering components.

Thus, the end customer requests are never routed to the down machine and the restart makes the system perform better! If restarting the machine is overkill, another option is to just restart the Oracle9*i*AS instance. If even that is an overkill, restarting individual OC4J instances is recommended, especially if you have deployed unproven or new Java applications.

13.10 Stock Spares and Have a Backup Schedule

Oracle9*i*AS provides easy commands to backup and restore configurations. However, these need to be executed periodically to gain the benefits of these! Moreover, in general Oracle9*i*AS will be one piece of a larger puzzle.

Hence, care should be taken that the backup and recovery operational schedule support restoring the entire Web site. This includes brand-new computers, storage, and network equipment with less than 24 hours combined downtime and data loss. This backup and spare parts storage provides a last line of defense against the worst failures, such as careless employees, botched upgrades, security break-ins, and software bugs that corrupt stored data.

If a failure requiring restoration of the database from a backup happens once a year and takes 24 hours to fix, then availability is 99.7%. It is possible to reduce the database restore time to a few minutes, using a physical standby database. With this scheme, the standby database is always in recovery mode, and its state lags the primary database by some fixed period (for example, 15 minutes). When the primary fails, applications switch to the standby. The hard part here is to detect the failure, which could be a corrupt infrequently accessed disk block, accidentally run batch job, or other non-obvious failure. It must be detected within the fixed period lag, before it possibly gets propagated to the standby.

It is important to backup everything, including router, firewall, and load balancer configuration, operating systems and their configuration, Oracle9*i*AS software and its configuration. But the backend Oracle database should be where most of your data is stored; it should receive the most attention.

A

Oracle9*i*AS Web Cache Best Practices Appendix

This appendix contains lengthier discussions of Oracle9*i*AS Web Cache best practices in the main sections of this document. It features the following topic:

A.1 Use Partial Page Caching Where Possible

Personalization is common in dynamic content. There are at least three common challenges when caching personalized content:

1. Many personalized pages cannot be cached for long or at all. Personalization often creates pages that consist of fragments with different caching properties. For example, a Portal page may include stock quotes that expire in 20 minutes, news that expires in three hours, and rotating ad banners that should not be cached. To serve consistent content, traditional full-page caches need to update the entire page at the highest change frequency of all its fragments.
2. The customizable combination of fragments creates a vast number of unique pages. Cache hit ratios will be low even if these unique pages are all cacheable.
3. Personalized information often appears in Web pages, making them unique for each user. For example, many Web pages contain tens or hundreds of hyperlinks embedding application session IDs.

To solve the first two challenges, Oracle9*i*AS Web Cache operates in a partial-page model, in which each Web page can be divided into a template and multiple fragments that can in turn be further divided into templates and lower level fragments. Each fragment or template is stored and managed independently; a full page is assembled from the underlying fragments upon request. Fragments can be shared among different templates, so that common fragments are not duplicated to waste precious cache space. Sharing can also greatly reduce the number of updates required when fragments expire. Depending on the application, updating a fragment can be cheaper than updating a full page. In addition, each template or fragment may have its own unique caching policies such as expiration, validation, and invalidation, so that each fragment in a full Web page can be cached as long as possible, even when some fragments are not cached or are cached for a much shorter period of time.

Oracle9*i*AS Web Cache uses ESI, to achieve flexible partial-page caching. ESI is a simple markup language for partial-page caching. Applications can mark up HTTP responses with two different kinds of tags, inline and include, that define the fragment/template structure in the response.

To address the third challenge in caching personalized content, Oracle9*i*AS Web Cache allows application developers to use variables in an ESI template. Because variables can be resolved to different pieces of request information or response information, the uniqueness of templates and fragments can be significantly reduced when personal information abounds.

There are two kinds of ESI variables: request variables and response variables. When an ESI template is assembled, a request variable is instantiated to a piece of request information such as a query string parameter, a cookie, or an HTTP header. For example, when a request for a dynamic page carries an application session ID in a query string parameter, this page may contain many hyperlinks with ESI request variables accessing this session ID, so that generated hyperlinks can carry the session ID into the next clicked page.

A response variable is similar to a request variable, except that its value comes not from the request, but from a special fragment called ESI environment. An ESI environment is essentially a special type of fragment whose response defines a set of variables that can be accessed by response variable occurrences in the enclosing template. For example, a dynamic page with a calendar may need to present personal appointments that cannot be stored in browser cookies due to cookie size limits. The application can instead reference a “profile” environment fragment in the template, and refer to all appointments in the environment without making separate requests for each appointment. In addition, an environment may be used to merge multiple small fragments into one environment by which each fragment can be referenced through response variable instantiation. This reduces storage and retrieval overhead similarly.

To encourage rapid adoption of ESI by Java developers, the creators of ESI have also published the Edge Side Includes for Java (JESI) specification. JESI is actively making its way through the Java Community Process standards body as JSR 128. As a specification and custom JSP tag library that developers can use to automatically generate ESI code, JESI facilitates the programming of Java Server Pages (JSPs) using ESI. While developers can always use ESI tags directly within their JSP code, JESI represents an easy way to express the modularity of JSPs and the caching of those modules, without requiring developers to learn a new programming syntax. JESI generates the appropriate ESI tags and headers in the JSP output that instruct ESI processors, such as Oracle9*i*AS Web Cache, to cache (or not) templates and fragments for the appropriate duration. JESI also facilitates the partial execution of JSPs when an ESI processor requests fragments and templates. Both OracleJSP (part of Oracle9iAS Containers for J2EE) and Oracle JDeveloper9i support the use of ESI and JESI, and both currently ship with the JESI tag library.

See Also:

- *Oracle9iAS Web Cache Administration and Deployment Guide*
- *Oracle9iAS Containers for J2EE JSP Tag Libraries and Utilities Reference*
- `http://www.esi.org`

A.2 Use <esi:inline> Tags for Existing Applications and <esi:include> Tags for New Applications

The `<esi:inline>` and `<esi:include>` tags enable applications to adopt ESI page fragmentation and assembly.

Review the following sections:

- Using Inline for Non-Fetchable Fragmentation
- Using Include for Fetchable Fragmentation

A.2.1 Using Inline for Non-Fetchable Fragmentation

Most existing applications are only designed to output an entire Web page to HTTP requests. These fragments and templates are non-fetchable, meaning they are not to be fetched independently from the origin server. If a cache needs any of these fragments or templates, the corresponding full Web page must be requested. To use ESI page assembly for non-fetchable fragments, an application can output the full-page response just as it does normally, with the exception that at the beginning and the end of each fragment, an `<esi:inline>` tag is inserted with a fragment name to demarcate the fragment. Oracle9*i*AS Web Cache stores the enclosed portions as separate fragments and the original page as page templates without the enclosed fragments. Fragments are shared if their names are identical.

When an application uses non-fetchable `<esi:inline>` fragments, the full page must be requested for every cache miss. At first, it can appear that there is no apparent cache benefit for cache misses. However, non-fetchable `<esi:inline>` fragments improves overall caching by:

- Increasing the cache-hit ratio because shared cacheable fragments can be extracted and stored only once, the size of the dynamic portion is reduced. A reduced space requirement results in a higher cache hit ratio than full page caching.
- Reducing cache update frequency Dynamic shared cacheable fragments require only one update. For example, a shared stock market fragment may expire much more frequently than any other parts of the page. With `<esi:inline>`fragmentation, only one cache update of any full page containing this fragment is enough to bring all full pages sharing this fragment current. Therefore, even non-fetchable `<esi:inline>` fragments can significantly reduce cache update frequency. The cost reduction is proportional to the degree of sharing.

`<esi:inline>` is primarily intended for pages with cacheable fragments. If a page contains non-cacheable, non-fetchable fragments, then the use of `<esi:inline>` is not recommended. However, the update of this full page may still offer benefit if it contains some cacheable fragments that are shared with other pages.

A.2.2 Using Include for Fetchable Fragmentation

The `<esi:include>` tag is another way to define fragments and templates in an HTTP output for dynamic content caching and assembly. It is in many ways similar to the <esi:inline> tag. It defines a name for the defined fragment.

The page including an `<esi:include>` tag is a template that references the defined fragment. However, it also has some key differences which makes its applicable scenarios very different from those of <esi:inline>:

- An `<esi:include>` tag in a template only defines the reference to a fragment.

  It does not enclose an embedded fragment directly in the template.

- A fragment referenced by an `<esi:include>` tag must always be independently fetchable by HTTP or HTTPS.

  The requested URL is the same as the fragment name. In contrast, an <esi:inline> tag's name only identifies the uniqueness of the fragment and is not used to fetch the actual content. The attribute defining the fragment name in `<esi:include>` fragment is src instead of name.

There are at least two scenarios where using `<esi:include>` tags is beneficial:

- Some applications, such as a Web portal, naturally assemble content from external sources. The application only provides a template that is used to fetch various fragments from third-party sources. In this case, the `<esi:include>` tags fetch and assemble directly, reducing one layer of redundancy.
- Some applications offer faster responses for template-only or fragment-only requests than full-page requests that use <esi:inline> tags. If `<esi:include>` is used for page fragmentation and assembly, Oracle9*i*AS Web Cache can miss only on the templates or fragments when most or all fragments are already cached, saving effective cache miss cost. In many cases, it is also valuable to cache the personalized templates because these seldom change.

A.3 Reduce Invalidation Overhead

When Oracle9*i*AS Web Cache receives an advanced invalidation request, it traverses the contents of the cache to locate the objects to invalid. Depending on the structure and number of objects cached, it can take time for Oracle9*i*AS Web Cache to invalid content. Here are some ways you can expedite cache content traversal:

- Send basic invalidation requests for invalidating one object.
- Use substring matching for invalidating multiple objects in advanced invalidations.

Also review the following sections:

- Send Basic Invalidation Requests for Invalidating One Object
- Use Substring Matching for Invalidating Multiple Objects in Advanced Invalidations

A.3.1 Send Basic Invalidation Requests for Invalidating One Object

When you need to invalidate one object in the cache, send a basic rather than an advanced invalidation request to avoid cache traversal.

To send a basic invalidation request, use the Basic Invalidation option in the Content Invalidation page (Administration > Content Invalidation) in Oracle9*i*AS Web Cache Manager or specify the `BASICSELECTOR` element in a manual invalidation request.

For example, the following request invalidates a document exactly matching /contacts/contacts.html using the `BASICSELECTOR` element:

```
<?xml version="1.0"?>
<!DOCTYPE INVALIDATION SYSTEM
"internal:///WCSinvalidation.dtd">
<INVALIDATION VERSION="WCS-1.1">
     <OBJECT>
              <BASICSELECTOR
URI="http://www.company.com:80/contacts/contacts.html"/>
         <ACTION REMOVALTTL="0"/>
     </OBJECT>
</INVALIDATION>
```

Advanced invalidation requests should be reserved for invalidation of multiple objects.

To send an advanced invalidation request, use the Advanced Invalidation option in the Content Invalidation page or specify the ADVANCEDSELECTOR element in a manual invalidation request.

A.3.2 Use Substring Matching for Invalidating Multiple Objects in Advanced Invalidations

When multiple objects share a common URL, request POST body, or an embedded URL parameter, you can express the common elements in multiple ways:

- Common URL:
  - Use the URL Regular Expression field in the Content Invalidation page.
  - Use the URIEXP attribute of the ADVANCEDSELECTOR element.
  - Use the ADVANCEDSELECTOR element's OTHER element with a NAME attribute value of URI.
- Common Request POST Body:
  - Use the HTTP Method and POST Body Expression fields in the Content Invalidation page
  - Use the ADVANCEDSELECTOR element's METHOD and BODYEXP attributes
  - Use the ADVANCEDSELECTOR element's OTHER element with a NAME attribute value of BODY.
- Common Embedded URL Parameter:
  - Use the ADVANCEDSELECTOR element's OTHER element with a NAME attribute value of QUERYSTRING_PARAMETER.

For the quickest invalidation, Oracle Corporation recommends using the OTHER element to specify a substring for literal matching rather than regular expression for pattern matching.

To send an advanced invalidation request with substring matching:

1. Specify the OTHER element in a manual invalidation request that uses the ADVANCEDSELECTOR element.
2. Specify the NAME attribute to use a value of URI, BODY, or QUERYSTRING_PARAMETER.
3. Specify the TYPE attribute to use a value of SUBSTRING.

For example, the following request searches for any documents underneath `http://wc-cluster.us.oracle.com:1100/pls/portal/!PORTAL.wwpro_app_provider.execute_portlet/595897563/`, that match the following criteria:

- The HTTP request method is an HTTP `POST` request method
- The URI is show`Portlet.Show`
- The HTTP `POST` body contains `_language=EN-US`
- The HTTP requests headers are `x-oracle-cache-user` and `x-oracle-cache-subid`
- The embedded URL parameters are `_portlet_id` and `_provider_id`

```
<?xml version="1.0"?>
<!DOCTYPE INVALIDATION SYSTEM
"http://www.oracle.com/webcache/90200/WCSinvalidation.dtd">
<INVALIDATION VERSION="WCS-1.1">
     <OBJECT>
         <ADVANCEDSELECTOR
URIPREFIX="/pls/portal/!PORTAL.wwpro_app_provider.execute_portl
et/595897563/"
HOST="wc-cluster.us.oracle.com:1100" METHOD="POST">
             <OTHER NAME="QUERYSTRING_PARAMETER" TYPE="SUBSTRING"
VALUE="_portlet_id=2"/>
             <OTHER NAME="QUERYSTRING_PARAMETER" TYPE="SUBSTRING"
VALUE="_provider_id=595897563"/>
            <HEADER NAME="x-oracle-cache-user" VALUE="PORTAL"/>
            <HEADER NAME="x-oracle-cache-subid" VALUE="1"/>
            <OTHER NAME="BODY" TYPE="SUBSTRING" VALUE="_language=EN-US"/>
            <OTHER NAME="URI" TYPE="SUBSTRING"
VALUE="showPortlet.Show"/>
            </ADVANCEDSELECTOR>
            <ACTION REMOVALTTL="0"/>
            <INFO VALUE="Invalidate an old portlet based on portlet ID
and
provider ID"/>
    </OBJECT>
</INVALIDATION>
```

See Also: *Oracle9iAS Web Cache Administration and Deployment Guide.*