



Sun Java™ System

Application Server 7 Migrating and Redeploying Server Applications Guide

2004Q2

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95054
U.S.A.

Part No: 817-5052

Copyright © 2004 Sun Microsystems, Inc., 4150 Network Circle, Santa Clara, California 95054, U.S.A. All rights reserved.

Sun Microsystems, Inc. has intellectual property rights relating to technology embodied in the product that is described in this document. In particular, and without limitation, these intellectual property rights may include one or more of the U.S. patents listed at <http://www.sun.com/patents> and one or more additional patents or pending patent applications in the U.S. and in other countries.

THIS PRODUCT CONTAINS CONFIDENTIAL INFORMATION AND TRADE SECRETS OF SUN MICROSYSTEMS, INC. USE, DISCLOSURE OR REPRODUCTION IS PROHIBITED WITHOUT THE PRIOR EXPRESS WRITTEN PERMISSION OF SUN MICROSYSTEMS, INC.

U.S. Government Rights - Commercial software. Government users are subject to the Sun Microsystems, Inc. standard license agreement and applicable provisions of the FAR and its supplements.

Use is subject to license terms. This distribution may include materials developed by third parties.

Sun, Sun Microsystems, the Sun logo, Java, Solaris, Sun™ ONE, Sun™ ONE Studio, iPlanet, J2EE, J2SE, Enterprise JavaBeans, EJB, JavaServer Pages, JSP, JDBC, JDK, JVM, Java Naming and Directory Interface, JavaMail, and the Java Coffee Cup logo are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. in the U.S. and other countries. Products bearing SPARC trademarks are based upon architecture developed by Sun Microsystems, Inc.

UNIX is a registered trademark in the U.S. and in other countries, exclusively licensed through X/Open Company, Ltd.

This product is covered and controlled by U.S. Export Control laws and may be subject to the export or import laws in other countries. Nuclear, missile, chemical biological weapons or nuclear maritime end uses or end users, whether direct or indirect, are strictly prohibited. Export or reexport to countries subject to U.S. embargo or to entities identified on U.S. export exclusion lists, including, but not limited to, the denied persons and specially designated nationals lists is strictly prohibited.

DOCUMENTATION IS PROVIDED "AS IS" AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.

Copyright © 2004 Sun Microsystems, Inc., 4150 Network Circle, Santa Clara, California 95054, Etats-Unis. Tous droits réservés.

Sun Microsystems, Inc. détient les droits de propriété intellectuelle relatifs à la technologie incorporée dans le produit qui est décrit dans ce document. En particulier, et ce sans limitation, ces droits de propriété intellectuelle peuvent inclure un ou plus des brevets américains listés à l'adresse <http://www.sun.com/patents> et un ou les brevets supplémentaires ou les applications de brevet en attente aux Etats-Unis et dans les autres pays.

CE PRODUIT CONTIENT DES INFORMATIONS CONFIDENTIELLES ET DES SECRETS COMMERCIAUX DE SUN MICROSYSTEMS, INC. SON UTILISATION, SA DIVULGATION ET SA REPRODUCTION SONT INTERDITES SANS L'AUTORISATION EXPRESSE, ECRITE ET PREALABLE DE SUN MICROSYSTEMS, INC.

L'utilisation est soumise aux termes de la Licence. Cette distribution peut comprendre des composants développés par des tierces parties.

Sun, Sun Microsystems, le logo Sun, Java, Solaris, Sun™ ONE, Sun™ ONE Studio, iPlanet, J2EE, J2SE, Enterprise JavaBeans, EJB, JavaServer Pages, JSP, JDBC, JDK, JVM, Java Naming and Directory Interface, JavaMail, et le logo Java Coffee Cup sont des marques de fabrique ou des marques déposées de Sun Microsystems, Inc. aux Etats-Unis et dans d'autres pays.

Toutes les marques SPARC sont utilisées sous licence et sont des marques de fabrique ou des marques déposées de SPARC International, Inc. aux Etats-Unis et dans d'autres pays. Les produits portant les marques SPARC sont basés sur une architecture développée par Sun Microsystems, Inc.

UNIX est une marque déposée aux Etats-Unis et dans d'autres pays et licenciée exclusivement par X/Open Company, Ltd.

Ce produit est soumis à la législation américaine en matière de contrôle des exportations et peut être soumis à la réglementation en vigueur dans d'autres pays dans le domaine des exportations et importations. Les utilisations, ou utilisateurs finaux, pour des armes nucléaires, des missiles, des armes biologiques et chimiques ou du nucléaire maritime, directement ou indirectement, sont strictement interdites. Les exportations ou réexportations vers les pays sous embargo américain, ou vers des entités figurant sur les listes d'exclusion d'exportation américaines, y compris, mais de manière non exhaustive, la liste de personnes qui font objet d'un ordre de ne pas participer, d'une façon directe ou indirecte, aux exportations des produits ou des services qui sont régis par la législation américaine en matière de contrôle des exportations et la liste de ressortissants spécifiquement désignés, sont rigoureusement interdites.

LA DOCUMENTATION EST FOURNIE "EN L'ETAT" ET TOUTES AUTRES CONDITIONS, DECLARATIONS ET GARANTIES EXPRESSES OU TACITES SONT FORMELLEMENT EXCLUES, DANS LA MESURE AUTORISEE PAR LA LOI APPLICABLE, Y COMPRIS NOTAMMENT TOUTE GARANTIE IMPLICITE RELATIVE A LA QUALITE MARCHANDE, A L'APTITUDE A UNE UTILISATION PARTICULIERE OU A L'ABSENCE DE CONTREFAÇON.

Contents

About This Guide	5
Who Should Use This Guide	5
Using the Documentation	6
How This Guide Is Organized	8
Documentation Conventions	9
General Conventions	9
Conventions Referring to Directories	11
Contacting Sun	11
Give Us Feedback	11
Obtain Training	11
Contact Product Support	12
 Chapter 1 Migration Considerations	13
Why Migrate?	13
Advantages in Sun Java™ System Application Server Standard and Enterprise 7 2004Q2	14
Developer Features	14
Operational Features	15
Additional Features in Sun Java System Application Server Enterprise	16
High Scalability Through Clustering	16
High Performance Through Load Balancing	16
High Availability Through Failover	17
Migration Strategy	17
What Needs to be Migrated?	17
Redeployment	18
 Chapter 2 Migrating to Sun Java System Application Server, Overview	21
Product Line Overview	21

Standard Edition	22
Enterprise Edition	22
Sun Java System Application Server Architecture	22
J2EE Component Standards	25
Development Environments	27
Application Server 6.0/6.5 Development Environment	27
Application Server 7 and 7 2004Q2 Development Environment	27
Administration Tools	29
Sun ONE Application Server 6.0 Administration Tools	29
Sun ONE Application Server 6.5 Administration Tools	30
Application Server 7 and 7 2004Q2 Administration Tools	30
Database Connectivity	32
J2EE Application Components and Migration	32
 Chapter 3 Migrating Applications to Sun Java System Application Server 7 2004Q2	35
About Sun ONE Application Server 6.0/6.5	36
Migration Issues From Sun ONE Application Server 6.x to Sun Java System Application Server 7 2004Q2	37
Migrating J2EE Components	38
Migrating JDBC Code	38
Establishing Connections Through the DriverManager Interface	39
Using JDBC 2.0 Data Sources	40
Migrating Java Server Pages and JSP Custom Tag Libraries	44
Migrating JSP Custom Tag Libraries	44
Migrating Servlets	45
Obtaining a Data Source from the JNDI Context	46
Declaring EJBs in the JNDI Context	46
EJB Migration	46
EJB Changes Specific to Sun Java System Application Server	47
Session Beans	47
Migrating EJB Applications that Support SFSB Failover (Enterprise Edition)	47
Entity Beans	49
Message Driven Beans	50
Migrating Web Applications	50
Migrating Web Application Modules	51
Particular setbacks when migrating servlets and JSPs	52
Migrating Enterprise EJB Modules	53
Migrating Applications to Support HTTP Failover (Enterprise Edition)	54
Migrating Enterprise Applications	57
Application Root Context and Access URL	58
Applications With Form-based Authentication	59
Migrating Proprietary Extensions	61
Migrating UIF	62

Approach 1: Checking in the registry files	62
Approach 2: Checking for UIF binaries in installation directories	63
Migration Process	64
Migrating Rich Clients	65
Authenticating a Client in 6.x	65
Authenticating a Client in 7 2004Q2 SE/EE	65
Using ACC in 6.x and 7 2004Q2 EE	65
Load-balancing and Failover Features in ACC Clients (Enterprise Edition)	66
Migrating Deployment Descriptors	67
Migrating Sun ONE Application Server 7 Applications	68
Migrating Rich Clients	69
Migrating Rich Clients From 7 PE/SE to 7 2004Q2 EE	69
Migrating Rich Clients From 7EE to 7 2004Q2EE	70
Migrating 7 2004Q2, SE Applications to Sun Java System Application Server 7 2004Q2, EE ..	71
Migrating EJB Applications to Support SFSB Failover	72
Migrating EJB Applications From 7.0SE/PE/EE to 7 2004Q2EE	72
 Chapter 4 Installation, Administration, and Deployment	73
Installation differences	73
Minimum Requirements	74
Installation Procedure differences	75
Administration and Deployment Differences	76
Non-root Installation and Administration	77
Deployment Topologies	77
 Chapter 5 Migrating iBank Application - Walkthrough	79
Preparing for Migrating the iBank Application	80
Manual Migration of iBank Application	82
Web Application Changes	82
EJB Changes	83
Assembling Application for Deployment	103
Deploying iBank application on Sun Java System Application Server 7 2004Q2 using the asadmin utility	104
 Appendix A iBank Application Specification	105
Tools used for the development of the application	106
Database schema	106
Application navigation and logic	110
Application Components	115
Fitness of design choices with regard to potential migration issues	118

Appendix B Migration Resources	121
Migrating Applications From Competitive Application Servers	121
Migrating from BEA WebLogic to Sun Java System Application Server 7	121
Migrating from IBM WebSphere to Sun Java System Application Server 7	122
Migration Tools	122
Sun ONE Studio, Release 5 Update 1	122
Sun Java System Migration Tool for Application Server	123
Sun ONE Migration Toolbox for Applogic and NetDynamics	123
J2EE Application Verification Kit	124
References	124
Migrating to Sun ONE Application Server 6.0	124
Migrating to Sun ONE Application Server 6.5	124
Migrating to Sun ONE Application Server 7	125
Redeploying Migrated Applications	125
 Appendix C Migrating from the Enterprise Java Beans 1.1 Specification to Enterprise Java Beans 2.0	 127
Differences Between EJB 1.1 and EJB 2.0	127
EJB Query Language	128
Local Interfaces	128
EJB 2.0 Container-Managed Persistence (CMP)	129
Defining Persistent Fields	130
Defining Entity Bean Relationships	130
Message-Driven Beans	131
Migrating EJB Client Applications	131
Declaring EJBs in the JNDI Context	131
Recap on Using EJB JNDI References	132
Placing EJB References in the JNDI Context	132
Global JNDI context versus local JNDI context	133
Migrating CMP Entity EJBs	133
Migrating the Bean Class	134
Migrating ejb-jar.xml	137
Custom Finder Methods	137

About This Guide

This *Migrating and Redeploying Server Applications Guide* describes how J2EE applications are migrated from earlier versions of Application Server to Sun Java™ System Application Server Standard and Enterprise Edition 7 2004Q2 environment.

This manual is intended for system administrators, network administrators, application server administrators and web developers who have an interest in migration issues.

This preface addresses the following topics:

- [Who Should Use This Guide](#)
- [Using the Documentation](#)
- [How This Guide Is Organized](#)
- [Documentation Conventions](#)
- [Contacting Sun](#)

Who Should Use This Guide

The intended audience for this guide is the person who develops, assembles, and deploys beans in a corporate enterprise.

This guide assumes you are familiar with the following topics:

- Java programming
- Java APIs as defined in the Java™ Servlet, JavaServer Pages™ (JSP™), Enterprise JavaBeans™ (EJB™), and Java™ Database Connectivity (JDBC™) specifications

- The SQL structured database query languages
- Relational database concepts
- Software development processes, including debugging and source code control

Using the Documentation

The Sun Java System Application Server Standard and Enterprise Edition manuals are available as online files in Portable Document Format (PDF) and Hypertext Markup Language (HTML).

The following table lists tasks and concepts described in the Sun Java System Application Server manuals. The manuals marked *(updated for 7 2004Q2)* have been updated for the Sun Java System Application Server Standard and Enterprise Edition 7 2004Q2 release. The manuals not marked in this way have not been updated since the version 7 Enterprise Edition release.

Table 1 Sun Java System Application Server Documentation Roadmap

For information about	See the following
<i>(Updated for 7 2004Q2)</i> Late-breaking information about the software and the documentation. Includes a comprehensive, table-based summary of supported hardware, operating system, JDK, and JDBC/RDBMS.	<i>Release Notes</i>
Sun Java System Application Server 7 overview, including the features available with each product edition.	<i>Product Overview</i>
Diagrams and descriptions of server architecture and the benefits of the Sun Java System Application Server architectural approach.	<i>Server Architecture</i>
<i>(Updated for 7 2004Q2)</i> How to get started with the Sun Java System Application Server product. Includes a sample application tutorial. There are two guides, one for Standard Edition and one for Enterprise Edition.	<i>Getting Started Guide</i>
<i>(Updated for 7 2004Q2)</i> Installing the Sun Java System Application Server Standard Edition and Enterprise Edition software and its components, such as sample applications and the Administration interface. For the Enterprise Edition software, instructions are provided for implementing the high-availability configuration.	<i>Installation Guide</i>
<i>(Updated for 7 2004Q2)</i> Evaluating your system needs and enterprise to ensure that you deploy Sun Java System Application Server in a manner that best suits your site. General issues and concerns that you must be aware of when deploying an application server are also discussed.	<i>System Deployment Guide</i>

Table 1 Sun Java System Application Server Documentation Roadmap (*Continued*)

For information about	See the following
Creating and implementing Java™ 2 Platform, Enterprise Edition (J2EE™ platform) applications intended to run on the Sun Java System Application Server that follow the open Java standards model for J2EE components such as servlets, Enterprise JavaBeans™ (EJBs™), and JavaServer Pages™ (JSPs™). Includes general information about application design, developer tools, security, assembly, deployment, debugging, and creating lifecycle modules. A comprehensive Sun Java System Application Server glossary is included.	<i>Developer's Guide</i>
(Updated for 7 2004Q2) Creating and implementing J2EE web applications that follow the Java™ Servlet and JavaServer Pages (JSP) specifications on the Sun Java System Application Server. Discusses web application programming concepts and tasks, and provides sample code, implementation tips, and reference material. Topics include results caching, JSP precompilation, session management, security, deployment, SHTML, and CGI.	<i>Developer's Guide to Web Applications</i>
(Updated for 7 2004Q2) Creating and implementing J2EE applications that follow the open Java standards model for enterprise beans on the Sun Java System Application Server. Discusses Enterprise JavaBeans (EJB) programming concepts and tasks, and provides sample code, implementation tips, and reference material. Topics include container-managed persistence, read-only beans, and the XML and DTD files associated with enterprise beans.	<i>Developer's Guide to Enterprise JavaBeans Technology</i>
(Updated for 7 2004Q2) Creating Application Client Container (ACC) clients that access J2EE applications on the Sun Java System Application Server.	<i>Developer's Guide to Clients</i>
Creating web services in the Sun Java System Application Server environment.	<i>Developer's Guide to Web Services</i>
(Updated for 7 2004Q2) Java™ Database Connectivity (JDBC™), transaction, Java Naming and Directory Interface™ (JNDI), Java™ Message Service (JMS), and JavaMail™ APIs.	<i>Developer's Guide to J2EE Services and APIs</i>
Creating custom NSAPI plug-ins.	<i>Developer's Guide to NSAPI</i>
(Updated for 7 2004Q2) Information and instructions on the configuration, management, and deployment of the Sun Java System Application Server subsystems and components, from both the Administration interface and the command-line interface. Topics include cluster management, the high-availability database, load balancing, and session persistence. A comprehensive Sun Java System Application Server glossary is included.	<i>Administration Guide</i>
(Updated for 7 2004Q2) Editing Sun Java System Application Server configuration files, such as the <code>server.xml</code> file.	<i>Administrator's Configuration File Reference</i>
Configuring and administering security for the Sun Java System Application Server operational environment. Includes information on general security, certificates, and SSL/TLS encryption. HTTP server-based security is also addressed.	<i>Administrator's Guide to Security</i>

Table 1 Sun Java System Application Server Documentation Roadmap (*Continued*)

For information about	See the following
Configuring and administering service provider implementation for J2EE™ Connector Architecture (CA) connectors for the Sun Java System Application Server. Topics include the Administration Tool, Pooling Monitor, deploying a JCA connector, and sample connectors and sample applications.	<i>J2EE CA Service Provider Implementation Administrator's Guide</i>
(Updated for 7 2004Q2) Migrating your applications to the new Sun Java System Application Server programming model, specifically from iPlanet Application Server 6.x and Sun ONE Application Server 7.0. Includes a sample migration.	<i>Migrating and Redeploying Server Applications Guide</i>
(Updated for 7 2004Q2) How and why to tune your Sun Java System Application Server to improve performance.	<i>Performance Tuning Guide</i>
(Updated for 7 2004Q2) Information on solving Sun Java System Application Server problems.	<i>Troubleshooting Guide</i>
(Updated for 7 2004Q2) Information on solving Sun Java System Application Server error messages.	<i>Error Message Reference</i>
(Updated for 7 2004Q2) Utility commands available with the Sun Java System Application Server; written in manpage style.	<i>Utility Reference Manual</i>
Using the Sun™ Java System Message Queue 3.5 software.	The Sun Java System Message Queue documentation at: http://docs.sun.com/db?p=prod/s1.slmsgqu

How This Guide Is Organized

This guide provides information on migrating applications to Sun Java System Application Server 7 2004Q2. The guide is organized as follows:

- [Chapter 1, “Migration Considerations,”](#) describes the enhancements available in Sun Java System Application Server 7 2004Q2.
- [Chapter 2, “Migrating to Sun Java System Application Server, Overview,”](#) describes the architecture of the Sun Java System Application Server 7 2004Q2 and the differences between J2EE standards and application components implemented with this version of the Sun Java System Application Server versus previous versions.

- [Chapter 3, “Migrating Applications to Sun Java System Application Server 7 2004Q2,”](#) describes considerations and strategies for migrating applications from Sun ONE Application Server 6.x, Sun ONE Application Server 7, to Sun Java System Application Server 7 2004Q2, Standard Edition and Enterprise Edition. There are also sample migration applications included that provide an end-to-end description of the migration process.
- [Chapter 4, “Installation, Administration, and Deployment,”](#) describes the difference between installing and administering Sun Java System Application Server 7 2004Q2 and Sun ONE Application Server 6.x Enterprise Edition. Also includes a brief description of the deployment topologies.
- [Chapter 5, “Migrating iBank Application - Walkthrough,”](#) describes the process for migrating the main components of a J2EE application to Sun Java System Application Server 7 2004Q2. Uses iBank sample application as the example to demonstrate the steps.
- [Appendix A, “iBank Application Specification,”](#) describes the iBank specification that is used throughout this guide.
- [Appendix B, “Migration Resources,”](#) describes the resources that helps in migrating the J2EE applications to Sun Java System Application Server 7 2004Q2.
- [Appendix C, “Migrating from the Enterprise Java Beans 1.1 Specification to Enterprise Java Beans 2.0,”](#) describes the modifications required to migration from EJB 1.1 specification to EJB 2.0.

Documentation Conventions

This section describes the types of conventions used throughout this guide:

- [General Conventions](#)
- [Conventions Referring to Directories](#)

General Conventions

The following general conventions are used in this guide:

- **File and directory paths** are given in UNIX® format (with forward slashes separating directory names). For Windows versions, the directory paths are the same, except that backslashes are used to separate directories.

- **URLs** are given in the format:

`http://server.domain/path/file.html`

In these URLs, *server* is the server name where applications are run; *domain* is your Internet domain name; *path* is the server's directory structure; and *file* is an individual filename. Italic items in URLs are placeholders.

- **Font conventions** include:
 - The `monospace` font is used for sample code and code listings, API and language elements (such as function names and class names), file names, pathnames, directory names, and HTML tags.
 - *Italic* type is used for code variables.
 - *Italic* type is also used for book titles, emphasis, variables and placeholders, and words used in the literal sense.
 - **Bold** type is used as either a paragraph lead-in or to indicate words used in the literal sense.
- **Installation root directories** for most platforms are indicated by *install_dir* in this document. Exceptions are noted in [“Conventions Referring to Directories” on page 11](#).

By default, the location of *install_dir* on **most** platforms is:

- Solaris and Linux file-based installations:

user's home directory/sun/appserver7

- Windows, all installations:

system drive: \Sun\AppServer7

For the platforms listed above, *default_config_dir* and *install_config_dir* are identical to *install_dir*. See [“Conventions Referring to Directories” on page 11](#) for exceptions and additional information.

- **Instance root directories** are indicated by *instance_dir* in this document, which is an abbreviation for the following:

default_config_dir/domains/domain/instance
- **UNIX-specific descriptions** throughout this manual apply to the Linux operating system as well, except where Linux is specifically mentioned.

Conventions Referring to Directories

By default, when using the Solaris package-based or Linux RPM-based installation, the application server files are spread across several root directories. This guide uses the following document conventions to correspond to the various default installation directories provided:

- *install_dir* refers to `/opt/SUNWappserver7`, which contains the static portion of the installation image. All utilities, executables, and libraries that make up the application server reside in this location.
- *default_config_dir* refers to `/var/opt/SUNWappserver7/domains`, which is the default location for any domains that are created.
- *install_config_dir* refers to `/etc/opt/SUNWappserver7/config`, which contains installation-wide configuration information such as licenses and the master list of administrative domains configured for this installation.

Contacting Sun

You might want to contact Sun Microsystems in order to:

- [Give Us Feedback](#)
- [Obtain Training](#)
- [Contact Product Support](#)

Give Us Feedback

If you have general feedback on the product or documentation, please send this to <http://www.sun.com/hwdocs/feedback>

Obtain Training

Application Server training courses are available at:

http://training.sun.com/US/catalog/enterprise/web_application.html/

Visit this site often for new course availability on the Sun Java System Application Server.

Contact Product Support

If you have problems with your system, contact customer support using one of the following mechanisms:

- The online support web site at:
<http://www.sun.com/supporttraining/>
- The telephone dispatch number associated with your maintenance contract

Please have the following information available prior to contacting support. This helps to ensure that our support staff can best assist you in resolving problems:

- Description of the problem, including the situation where the problem occurs and its impact on your operation
- Machine type, operating system version, and product version, including any patches and other software that might be affecting the problem. Here are some of the commonly used commands:
 - **Solaris:** `pkginfo, showrev`
 - **Linux:** `rpm`
 - **All:** `asadmin version --verbose`
- Detailed steps on the methods you have used to reproduce the problem
- Any error logs or core dumps
- Configuration files such as:
 - `instance_dir/config/server.xml`
 - a web application's `web.xml` file, when a web application is involved in the problem
- For an application, whether the problem appears when it is running in a cluster or standalone

Migration Considerations

This chapter provides an overview of the enhancements available in Sun Java™ System Application Server 7 2004Q2.

Why Migrate?

Sun Java System Application Server incorporates the latest Java technologies in an easy-to-use, developer-friendly package. The Application Server product leverages over six years of Sun expertise in delivering highly scalable application server technology, enabling developers to rapidly build robust applications that are based on JavaServer Pages (JSP™) technology, Java™ Servlet, and Enterprise JavaBeans™ (EJB™) technology. This technology supports a broad range of business requirements from small departmental applications to enterprise-scale, mission-critical services.

Although J2EE specifications broadly cover requirements for applications, it is nonetheless an evolving standard. It either does not cover some aspects of applications or leaves implementation details as the responsibility of application providers.

These product implementation-dependent aspects manifest as differences in the way application servers are configured and also in the deployment of J2EE components on application servers. The array of available configuration and deployment tools for use with any particular application server product also contribute to the product implementation differences.

The evolutionary nature of the specifications itself presents challenges to application providers. Each of the component APIs in turn are separately evolving. This leads to a varying degree of conformance by products. In particular, an emerging product such as Sun Java System Application Server, has to contend with

differences in J2EE application components, modules, and files deployed on other established application server platforms. Such differences require mappings between earlier implementation details of the J2EE standard such as file naming conventions, messaging syntax, and so forth.

Moreover, product providers usually bundle additional features and services with their products. These features are available as custom JSP tags or proprietary Java API libraries. Usage of such proprietary features render these applications non-portable.

Advantages in Sun Java™ System Application Server Standard and Enterprise 7 2004Q2

Sun Java System Application Server is an enhanced product of Sun ONE Application Server 7. The product introduces a variety of new features that enhance both the developer and operational experience. Like Sun ONE Application Server 7, Sun Java System Application Server 7 2004Q2 is J2EE 1.3 specification compliant.

The following new features are included in all editions of the product.

- [Developer Features](#)
- [Operational Features](#)
- [Additional Features in Sun Java System Application Server Enterprise](#)

Developer Features

The Application Server 7 2004Q2 distribution includes the following developer features:

- Java 2 Enterprise Edition 1.3 Compatible including
 - JavaServer Pages (JSP) 1.2 and Servlet 2.3 Support
 - Enterprise JavaBeans (EJB) 2.0 technology
 - Message Driven Beans (MDBs)
- Java 2 Platform, Standard Edition (J2SE™ platform) 1.4
- Integrated Java Web Services

- Updated Sun ONE Studio integration with seamless debugging and deployment
- Dynamic (“Hot”) Deployment and Reloading
- JSP Source-level Debugging
- Greatly Enhanced Container Managed Persistence (CMP) Support
- Easy-to-configure, XML-based Server Configuration
- Lifecycle Listener Classes (sophisticated startup and shutdown classes)
- Integrated J2EE Application Verification Utility
- Extensive Sample Applications
- Ant Build Facility Integration
- Easy Installation with Minimal Dependencies (no separate web server or directory server required)

Operational Features

The Application Server 7 2004Q2 distribution includes the following operational features:

- Integrated, High-Performance HTTP Server
- Integrated, Proven Sun Java System Message Queue 3.5 (formerly called Sun ONE Message Queue), Platform Edition as the JMS Provider
- Virtual HTTP Server Support for Web Applications
- Multiple Administration Domains per Install Image (Separate application server configurations from single install image.)
- Web-based Administration
- Proxy Plug-in for redirecting requests from web server to application server
- Full-featured, Remotable Command-line Interface Supporting Remote Monitoring
- Pluggable Authentication Based on Java Authentication and Authorization Service (JAAS)
- Improved Logging with multiple log levels
- Platform-specific packaging with SVR4 style packages for Solaris

- Remote web-based Administration Interface
- Rights to use bundled Sun Java System Directory Server for User Authentication and limited application configuration
- Load Distribution by redirecting URIs configured on the web server to a different application server
- Remote monitoring using the SNMP monitoring facilities

Additional Features in Sun Java System Application Server Enterprise

Sun Java System Application Server Enterprise provides the following features over the standard edition:

- [High Scalability Through Clustering](#)
- [High Performance Through Load Balancing](#)
- [High Availability Through Failover](#)

High Scalability Through Clustering

You can add multiple instances of the Application Server to a single machine, thereby increasing the capacity of the system without degrading performance. Application Server instances can be distributed over many machines and can be grouped together in 'clusters' for easy manageability.

High Performance Through Load Balancing

Dynamic load balancing of the various instances or individual instances within clusters, ensures optimum performance of the Application Server and your J2EE applications. Since the load balancing configuration can be dynamically reloaded, there will be no disruption in service as you add more instances to a cluster.

High Availability Through Failover

Sun Java System Application Server Enterprise provides a highly available and reliable solution through the use of load balancing and a sophisticated failover mechanism.

In addition, the bundled High Availability Database ensures that HTTP/S session information are securely stored.

These enhanced features, along with the High-Availability Database server, lets you provision for a 24x7x365 service for your J2EE applications.

NOTE	Your rights to use the features are governed by License that you accept when installing the product. Please review the Supplemental Terms to determine the functionality you may use. Sun Java System Application Server 7 2004Q2 is an enhanced application server offering from Sun. This product does not support the AppLogic style applications supported by the earlier versions of Sun's Application Server.
-------------	---

Migration Strategy

This section discusses the application components that needs to be migrated. The J2EE components can be easily migrated using the Migration Tool. However, if you wish to migrate the components manually, you can follow the procedure outlined in the [Chapter 3, "Migrating Applications to Sun Java System Application Server 7 2004Q2."](#)

For a detailed migration walk through of the migration process, see [Chapter 5, "Migrating iBank Application - Walkthrough."](#) For planning your hardware and software requirements, see [Chapter 4, "Installation, Administration, and Deployment."](#)

What Needs to be Migrated?

For migration purposes, the J2EE application consists of the following file categories:

- [Deployment descriptors \(XML files\)](#)
- [JSP Source Files Containing Proprietary API's](#)

- [Java Source Files Containing Proprietary API's](#)

Deployment descriptors (XML files)

Deployment is accomplished by specifying deployment descriptors (DDs) for EJBs (ejb-jar), front-end web components (war) and enterprise applications (ear). Deployment descriptors are used to resolve all external dependencies of the J2EE components/applications. The J2EE specification for DDs is common across all application server products. However, the specification leaves several deployment aspects of components pertaining to an application dependent on product-implementation.

JSP Source Files Containing Proprietary API's

J2EE specifies how to extend JSP by adding extra custom tags. Product vendors include some custom JSP extensions in their products, simplifying some tasks for developers. However, usage of these proprietary *custom tags* results in non-portability of JSP files. Additionally, JSP can invoke methods defined in other Java source files as well. The JSP's containing proprietary API's needs to be rewritten before they can be migrated.

Java Source Files Containing Proprietary API's

The Java source files can be Servlets, EJBs or other helper classes. The Servlets and EJBs can invoke standard J2EE services directly. They can also invoke methods defined in helper classes. Java source files are used to encode the business layer of applications such as EJBs. Vendors bundle several services and proprietary Java API with their products. The usage of *proprietary Java API* is the major source of non-portability in applications. Since J2EE is an evolving standard, different products may support *different versions of J2EE component APIs*. This is another aspect that migration will address.

Files within the above file categories need to be migrated to Sun Java System Application Server. The details on how to migrate each of the indicated file categories are provided in [“Migration Issues From Sun ONE Application Server 6.x to Sun Java System Application Server 7 2004Q2”](#) on page 37.

Redeployment

Redeployment refers to deploying a previously deployed application from an earlier version of Sun ONE Application Server, or from applications that were previously deployed, but migrated, from a competing application server platform.

The act of redeploying an application typically refers to using the standard deployment actions outlined in the *Sun Java System Application Server Administration Guide*.

Migrating to Sun Java System Application Server, Overview

This chapter provides an overview of the Application Server 7 2004Q2 product line, and describes the architecture of the Sun Java™ Application Server 7 2004Q2 and the J2EE components that are integral to the server environment. In addition, the differences between the Sun Java System Application Server 7 2004Q2 product line environment and earlier Sun ONE Application Server environments are described.

The following topics are addressed:

- [Product Line Overview](#)
- [Sun Java System Application Server Architecture](#)
- [J2EE Component Standards](#)
- [Development Environments](#)
- [Administration Tools](#)
- [Database Connectivity](#)
- [J2EE Application Components and Migration](#)

Product Line Overview

Sun Java System Application Server 7 2004Q2 is an enhancement to Sun ONE Application Server 7. It incorporates the latest Java technologies in an easy-to-use, developer-friendly package. The Application Server product leverages over six years of Sun expertise in delivering highly scalable application server technology, enabling developers to rapidly build robust applications that are based on JavaServer Pages (JSP™) technology, Java™ Servlet, and Enterprise JavaBeans™

(EJB™) technology. This technology supports a broad range of business requirements from small departmental applications to enterprise-scale, mission-critical services. Two editions of the application server are offered to suit a variety of needs for both production and development environments:

- [Standard Edition](#)
- [Enterprise Edition](#)

Standard Edition

The Standard Edition offers the best platform for the developers. Standard Edition supports configuration of multiple application server instances per administrative domain. Additionally, you can use the Simple Network Monitoring Protocol (SNMP) to monitor your Standard Edition application server. Sun Java System Directory Server provides user authentication and limited application configuration storage.

Enterprise Edition

Enterprise Edition enhances the core application server platform with high availability, load balancing, and cluster management capabilities suited for the most demanding J2EE-based application deployments. The management capabilities of the Standard Edition are extended in Enterprise Edition to account for multiple-instance deployments.

Clustering support includes groups of cloned application server instances to which client requests can be load balanced. Both the web tier Load Balancing Plug-in and third-party hardware load balancers are supported by this edition. The patented “Always On,” High-Availability Database technology forms the basis for the high availability persistence store in the Enterprise Edition.

Sun Java System Application Server Architecture

Application servers provide the framework for a client to connect to a backend source, execute the application logic, and return the result to the client. The application server occupies the middle-tier in the three-tier computing model.

The Sun Java System Application Server 7 2004Q2 is an enhanced release of Sun ONE Application Server 7. This product line provides a robust J2EE platform for the development, deployment, and management of e-commerce application services to a broad range of servers, clients, and devices.

The key features of the Sun Java System Application Server 7 2004Q2 Enterprise Edition architecture includes:

- **Clustering:** a cluster is typically a group of Application Server instances configured to store session data to a single persistent datastore, the High-Availability Database, and a web server configured to use the load balancing plug-in. The cluster appears to external clients as a single installation of the Application Server. Individual server instances are referred to as nodes in a cluster.

For more information on setting up clustering, see chapter, “Cluster Management,” in *Sun Java System Application Server Administration Guide*.

- **Load Balancing:** the load balancing plug-in is an extension to the web server. The web server is configured to hand off responding to certain HTTP requests to the plug-in, which then distributes the HTTP requests across the individual nodes of the cluster.

For more information on setting up load balancing, see chapter, “Configuring Load Balancing,” in *Sun Java System Application Server Administration Guide*.

- **Session Persistence Failover:** when a node in a cluster that has been configured to persist session data fails, another node accesses the session of the failed node and continues to respond to the client. The session data is stored to the High-Availability Database (HADB), a transactional, highly available, and highly scalable data store.

For more information on setting up session persistence using the High-Availability Database, see chapters, “Configuring the High-Availability Database,” and “Session Persistence,” in *Sun Java System Application Server Administration Guide*.

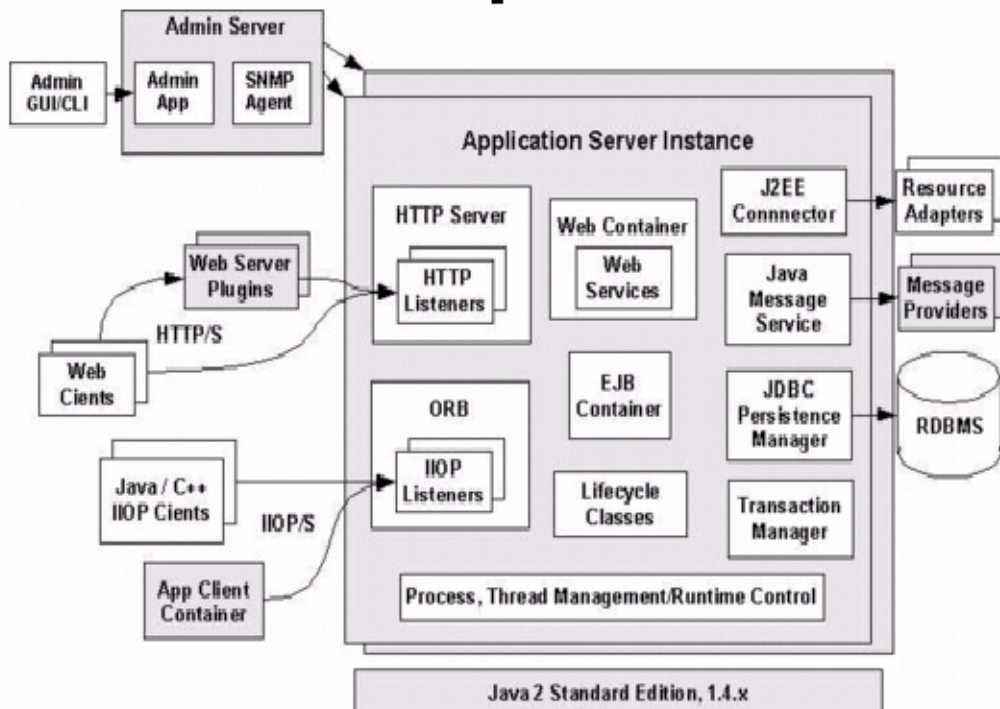
- **High Availability of EJBs along RMI/IIOP Path:** The load balancing and failover features along RMI/IIOP path routes the EJB method invocations along this path to one of the redundant instances when the server hosting the EJB becomes unavailable. HADB is used for storing the stateful session beans (SFSB) state for high availability.

For more information on configuring high availability along RMI/IIOP path using the High-Availability Database, see chapters, “Configuring the High-Availability Database,” and “Configuring CORBA Clients” in *Sun Java System Application Server Administration Guide*.

- High Availability of JMS Connections:** Sun Java System Application Server 7 2004Q2 is certified with Sun Java System Message Queue (MQ) 3.5 (formerly called Sun ONE Message Queue). MQ3.5 supports clustering and failover of JMS connections. That is, a JMS connection obtained for a given broker will be transparently failed-over to a new broker.

For more information about the high availability of JMS connections, see chapters, “JMS Configuration,” in *Sun Java System Application Server Administration Guide*.

Figure 2-1 Sun Java System Application Server Architecture



The Application Server architecture illustrated in the above diagram shows the Application Server component architecture, sub-systems, access paths, and how external entities interface with the core server.

Sun Java System Application Server 7 2004Q2 architecture, is highly componentized which results in a very highly manageable architecture. All the services required by the J2EE specification are present with well-defined standard interfaces to invoke them from within applications.

The web user interface provides for easy remote server management. In fact, the server is designed such that one administration server can be used to administer multiple numbers of administered servers.

By using the JDK 1.4 for the server operation, the Application Server utilizes the enhanced abilities of this newer version of JDK to its advantage.

A typical J2EE application is composed of an n -tier system in which a client obtains processed information from a Web server or an application server. The servers in turn access the information from enterprise systems such as RDBMS or ERP, process them by using contained business logic, and deliver the processed information to the client in an appropriate format. These layers can be designated as client layer (Web browser or rich Java client), middle layer (Web servers and application servers), and the back-end layer or data layer (enterprise systems such as databases).

The J2EE application model within the Sun Java System Application Server allows developers to focus on the business logic while J2EE components handle all the low level details. Therefore, applications and services can be easily enhanced and rapidly deployed, allowing business to quickly react to competitive changes. By providing an open standard architecture through the J2EE Platform, Sun Java System Application Server solves the problem of the cost and complexity in developing multi-tiered services that are scalable, highly available, secure and reliable.

J2EE Component Standards

Sun Java System Application Server 7 2004Q2 is a J2EE 1.3 compliant server based on the component standards developed by the Java community for Servlets, Java Server Pages (JSPs), and Enterprise JavaBeans (EJBs).

The following table characterizes the differences between the component APIs used with the J2EE 1.3 compliant Application Server 7 and 7 2004Q2 Enterprise and Standard editions, and the J2EE 1.2 Sun ONE Application Server 6.0/6.5.

Table 2-1 Application Server Version Comparison of APIs for J2EE Components

Component API	Sun ONE Application Server 6.0/6.5	Sun ONE Application Server 7 and Sun Java System application Server Standard and Enterprise 7 2004Q2
JDK	Sun 1.2.2	Sun 1.4
Servlet	2.2	2.3
JSP	1.1	1.2
JDBC	2.0	2.0
EJB	1.1	2.0
JNDI	1.2	1.2
JMS	1.0	1.1
JTA	1.0	1.01

In addition, the two products support a number of technologies connected with XML standards and Web Services which, while not part of the J2EE specification, are mentioned in the following table due to the increasing usage of these standards in enterprise applications.

Table 2-2 Additional Application Server Supported Technologies

Technology	Sun ONE Application Server 6.0/6.5	Sun ONE Application Server 7 and Sun Java System Application Server Standard and Enterprise 7 2004Q2
XML document processing (API and XML parser)	JAXP 1.0, Apache Xerces	JAXP 1.1
SOAP/Java support for Web Services	SOAP 1.1 (IBM SOAP4J framework)	Apache SOAP 2.2, JAX-RPC 1.1, JAXM 1.1, JAXR 1.0

Development Environments

This section characterizes the differences between the development environments for the Sun ONE Application Server 6.0/6.5 and the Sun Java System Application Server 7 2004Q2. The following topics are described:

- [Application Server 6.0/6.5 Development Environment](#)
- [Application Server 7 and 7 2004Q2 Development Environment](#)

Application Server 6.0/6.5 Development Environment

Sun ONE Application Server 6.0/6.5 offers an evaluation version of Sun ONE Studio for Java, which is especially geared towards application development for this version of the Sun ONE Application Server.

It is a very complete development environment in Java, based on the NetBeans platform. This IDE provides an extremely rich range of features for designing and developing Java applications and EJB components. It also integrates through a plug-in with Sun ONE Application Server for assembly, deployment, and debugging of the various J2EE components of an application. It is available in both Windows and Solaris.

Of the third-party vendor solutions available on the market, the recently released *Borland JBuilder 6 Enterprise* is an extremely mature, comprehensive product, with the added advantage of being available on several platforms (Windows, Solaris, Linux, and MacOS X). In addition to its Java development features (servlets, JSP pages, EJB components, graphic applications), JBuilder also caters for UML design, unit testing, collaborative development, and XML development. Moreover, JBuilder integrates perfectly with mainstream application servers (including the Sun ONE Application Server) for assembly, deployment and debugging of Web applications and EJB components.

Application Server 7 and 7 2004Q2 Development Environment

The availability of a fully integrated development solution is key to the success of the Sun Java System Application Server. Sun ONE Studio for Java Enterprise Edition 5, Update 1 is the strategic tool for Sun Java System application development.

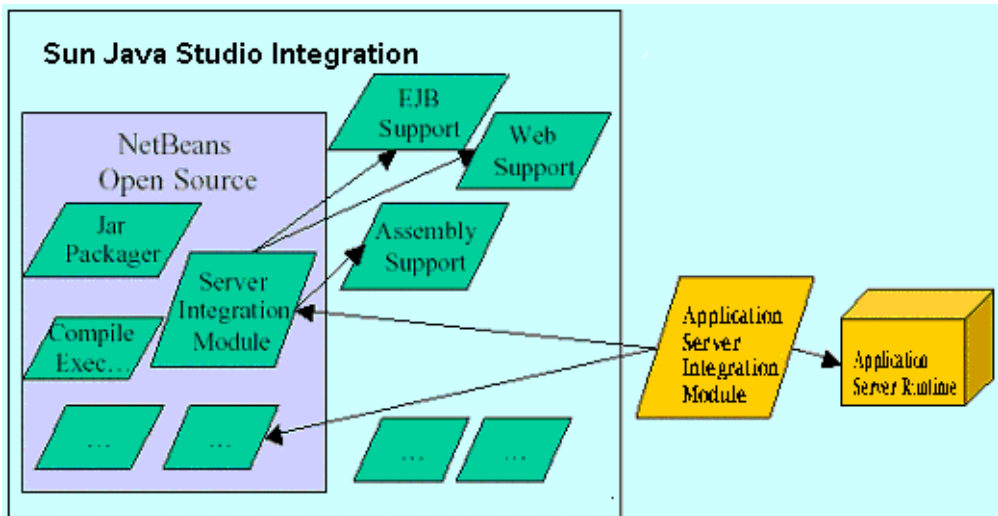
Sun ONE Studio for Java 5, update 1 is supported with Sun Java System Application Server 7 2004Q2.

Some of the key features of Sun ONE Studio Enterprise Edition 5 are:

- Ability to build EJBs quickly and easily
- Ability to assemble applications from EJBs and package applications for deployment
- Application server integration for deployment
- Ability to develop and publish web services
- Sun ONE Studio enterprise service presentation toolkit
- Parallel development using Sun ONE Studio Code Management Software
- Ability to integrate with the Sun Java System Application Server

As shown in the following figure, the Sun Java System Application Server integration module relies upon the NetBeans Open Source modules that are implemented from the Sun ONE Studio Close Source.

Figure 2-2 Sun ONE Studio 5 Update 1, Enterprise Edition and Sun Java System Application Server 7 2004Q2 Integration



Administration Tools

This section characterizes the differences between the administration tools for the Sun ONE Application Server 6.0, Sun ONE Application Server 6.5, and the Sun ONE Application Server 7. The following topics are described:

- [Sun ONE Application Server 6.0 Administration Tools](#)
- [Sun ONE Application Server 6.5 Administration Tools](#)
- [Application Server 7 and 7 2004Q2 Administration Tools](#)

Sun ONE Application Server 6.0 Administration Tools

Sun ONE Application Server 6.0 features a full set of graphical administration tools, which cover all the aspects of server management and administration

- **Sun ONE Console** - the main administration control panel. Sun ONE console gives fast access to the Administration Server Console, the Directory Server, and the Administration Tool.
- **Administration Server Console** - used to define event-logging options and to create SSL security certificates.
- **Sun ONE Directory Server Console** - used for administration of the Sun ONE Directory Server. The Directory Server is used to administer the two main information directory trees, the *user directory* (user and organizational unit administration), and the *configuration directory* (server configuration).
- **Sun ONE Administration Tool** - used to administer one or more instances of Sun ONE Application Server 6.0, along with the applications deployed. It also enables JDBC drivers and data sources to be configured.
- **Sun ONE Registry Editor** (*kregedit*) - is a graphical tool similar to the windows registry editor (*regedit*). It is used to adjust certain parameters specific to the Sun ONE Application Server, stored in a specific registry.

Sun ONE Application Server 6.5 Administration Tools

Sun ONE Application Server 6.5 can be administered using integrated Administration Tool, Sun ONE registry editor and command line tools, which are described below:

- **Sun ONE Application Server Administration Tool** - a stand-alone Java application with a graphical user interface that allows you to administer one or more instances of Sun ONE Application Server along with administering application components.
- **Command line tools** - can be run from the command-line prompt on Windows and the shell prompt on Solaris. You can perform a variety of tasks using the command line tools, right from basic configuration to deploying an application. To get a complete description of any command-line tool, type `[command] -help` at the command prompt. For ease of use, most of the command-line tools have been integrated with the Sun ONE Application Server Administration Tool and the Sun ONE Application Server Deployment Tool.
- **Sun ONE Registry Editor (*kregedit*)** - a stand-alone GUI tool similar to the Windows Registry editor (*regedit*). It can display and edit registry information for Sun ONE Application Server.

Application Server 7 and 7 2004Q2 Administration Tools

The Administration Server in is a special instance of the Server that serves the Administration interface and controls some global settings common to all server instances. It is a web-based server that contains the forms used to configure the Sun Java System Application Server.

This graphical tool allows you to manage your application server including viewing error and access logs, monitoring server usage, creating and editing virtual servers, apply configuration changes and start or stop server instances.

When you installed the Application Server, you chose a port number for the Administration Server, or used the default port of 4848. To access the Administration interface, in a web browser type:

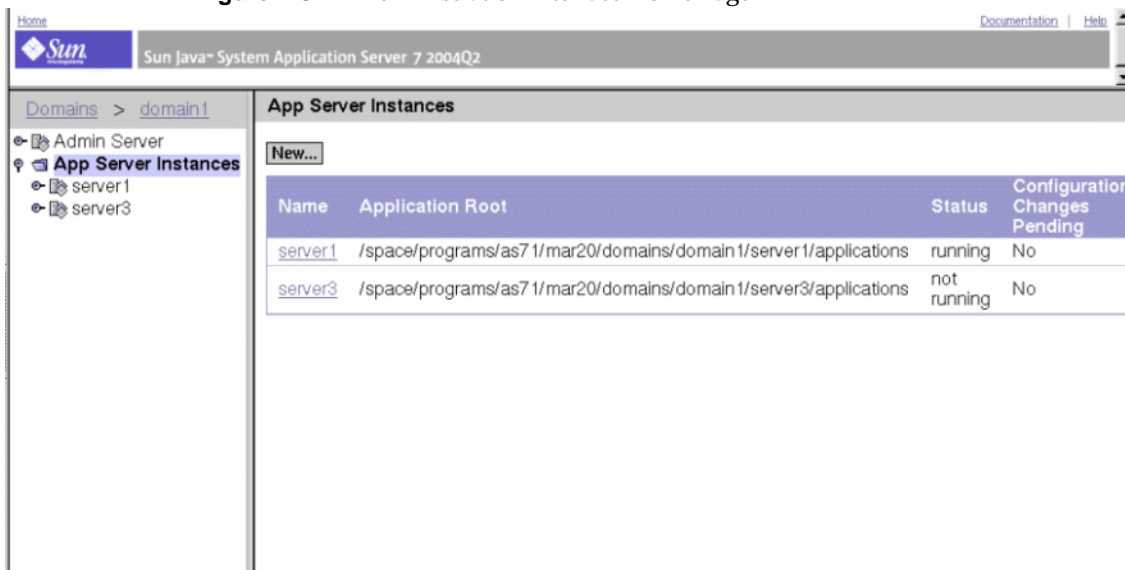
`http://hostname:port/admin`

You are prompted for the configured user name and password. Upon entering this information and clicking the OK button, the home page of the Administration interface is displayed, as shown in the following figure.

The left pane is a tree view of all items you can configure in the Sun Java System Application Server. To use the Administration interface, click an item in the left pane. The right pane displays the page associated with that item.

You can access help for any page in the Administration interface by clicking the Help button in the banner at the top of the Administration interface. The online help describes the use of the page you are accessing and gives information about what to enter in the fields on the page.

Figure 2-3 Administration Interface Home Page



Sun Java System Application Server contains a command line interface. You can use a utility and commands to perform the same set of tasks as you can perform in the Administration interface. You can use these commands either from a command prompt in the shell, or you can call them from other scripts and programs. Using these commands you can automate administration tasks that otherwise might become repetitive.

Sun Java System Application Server has a command line utility *asadmin*, which can be run from command-line prompt on Windows and the shell prompt on Solaris. The *asadmin* utility has a set of commands used to perform administrative tasks. You can use these commands to perform all the same tasks that are performed from the Administration Interface, from basic configuration to deploying an application. To get a complete description of any command, type *help* after entering the *asadmin* utility.

You can run *asadmin* either in single mode or multi-mode. In single mode you run one command at a time from the command prompt. In multi-mode you can run multiple commands without needing to reenter environment-level information.

Database Connectivity

See the *Sun Java System Application Server Installation Guide* for a complete description of the databases supported with each release of the Application Server.

J2EE Application Components and Migration

J2EE simplifies development of enterprise applications by basing them on standardized, modular components, providing a complete set of services to those components, and handling many details of application behavior automatically, without complex programming. J2EE 1.3 architecture includes several component APIs. Prominent J2EE APIs include:

- Servlets
- Java Server Pages (JSPs)
- EJBs, including Message Driven Beans (MDBs)
- Java Database Connectivity (JDBC)
- Java Transaction Service (JTS)
- Java Naming and Directory Interface (JNDI)
- Java Message Service (JMS)

J2EE components are packaged separately and bundled into a J2EE application for deployment. Each component, its related files such as GIF and HTML files or server-side utility classes, and a deployment descriptor are assembled into a module and added to the J2EE application. A J2EE application is composed of one or more enterprise bean(s), Web, or application client component modules. The final enterprise solution can use one J2EE application or be made up of two or more J2EE applications, depending on design requirements.

A J2EE application and each of its modules has its own deployment descriptor. A deployment descriptor is an XML document with a `.xml` extension that describes a component's deployment settings. An enterprise bean module deployment descriptor, for example, declares transaction attributes and security authorizations for an enterprise bean. Because deployment descriptor information is declarative, it can be changed without modifying the bean source code. At run time, the J2EE server reads the deployment descriptor and acts upon the component accordingly.

A J2EE application with all of its modules is delivered in an Enterprise Archive (EAR) file. An EAR file is a standard Java Archive (JAR) file with an `.ear` extension. The EAR file contains EJB JAR files, application client JAR files and/or Web Archive (WAR) files. The characteristics of these files are as follows:

- Each EJB JAR file contains a deployment descriptor, the enterprise bean files, and related files
- Each application client JAR file contains a deployment descriptor, the class files for the application client, and related files
- Each WAR file contains a deployment descriptor, the Web component files, and related resources

Using modules and EAR files makes it possible to assemble a number of different J2EE applications using some of the same components. No extra coding is needed; it is just a matter of assembling various J2EE modules into J2EE EAR files.

The migration process is concerned with moving J2EE application components, modules, and files.

For more information on migrating various J2EE components refer to [Chapter 3, "Migrating Applications to Sun Java System Application Server 7 2004Q2."](#)

For more background information on J2EE, see the following references:

- J2EE tutorial - <http://java.sun.com/j2ee/tutorial/>
- J2EE overview - <http://java.sun.com/j2ee/overview.html>
- J2EE topics - <http://java.sun.com/j2ee/>

Migrating Applications to Sun Java System Application Server 7 2004Q2

This chapter describes the considerations and strategies that are needed when moving J2EE applications from Sun ONE Application Server 6, 6.5, and 7 to the Sun Java System Application Server 7 2004Q2 product line.

This chapter also describes specific migration tasks at the component level.

NOTE	You are encouraged to use the Migration Tool for Application Servers that automates the migration of J2EE components to Sun Java System Application Server. For more information on the Migration Guide, see section “ Sun Java System Migration Tool for Application Server ” of Appendix B, “Migration Resources.”
-------------	--

The following topics are addressed:

- [About Sun ONE Application Server 6.0/6.5](#)
- [Migration Issues From Sun ONE Application Server 6.x to Sun Java System Application Server 7 2004Q2](#)
- [Migrating J2EE Components](#)
- [Migrating Web Applications](#)
- [Migrating Enterprise EJB Modules](#)
- [Migrating Enterprise Applications](#)
- [Migrating Proprietary Extensions](#)
- [Migrating UIF](#)

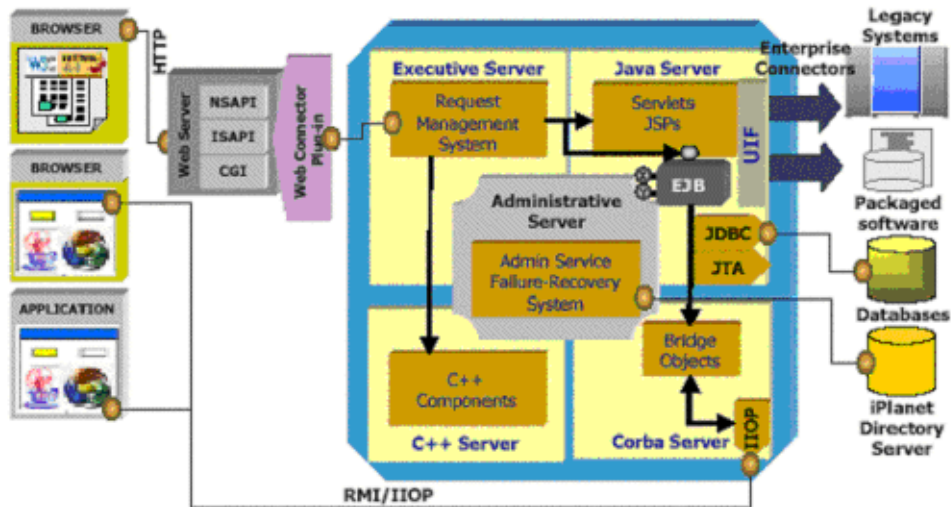
About Sun ONE Application Server 6.0/6.5

Sun ONE Application Server version 6.0 is a multi-platform application server based entirely on the J2EE 1.2 specification. Supported platforms include Windows NT and 2000, Solaris, AIX, and HP-UX.

In addition, Sun ONE Application Server 6.0 integrates with many Web servers through specific Web connector plug-ins that it ships with. These connectors enable it to be coupled with Sun ONE Web Server, Microsoft IIS, or Apache.

The Sun ONE Application Server 6.0/6.5 architecture is shown in the following figure:

Figure 3-1 Sun ONE Application Server 6.0/6.5 Architecture



As shown in the above figure, there are four internal servers, which are often called engines or processes. These processes are responsible for all the processing in the Sun ONE Application Server. The four internal servers of the Sun ONE Application Server 6.0/6.5 are:

Executive Server - provides most system services (some services are managed by the Administrative Server).

Administrative Server - provides system services for Sun ONE Application Server Administration and failure recovery.

Java Server - provides services to java applications.

C++ Server - components written in C++ are hosted in C++ server.

When a web server forwards requests to Sun ONE Application Server 6.0/6.5, the requests are first received by the Executive Server process (KXS). The KXS process forwards the request either to a Java Server process (KJS) or to a C++ Server process (KCS). A KJS process runs Java programming logic, whereas a KCS process runs C++ programming logic. Each KJS and KCS process maintains a specified number of threads and runs the programming logic to completion on those threads. The results are returned to the web server and sent on to the client browser.

Migration Issues From Sun ONE Application Server 6.x to Sun Java System Application Server 7 2004Q2

This section describes the issues that will arise while migrating the main components of a typical J2EE application from Sun ONE Application Server 6.0 and 6.5 to Sun Java System Application Server 7 2004Q2.

The migration issues described in this section are based on an actual migration that was performed for a J2EE application called *iBank*, a simulated online banking service, from Sun ONE Application Server 6.0 and 6.5 to Sun Java System Application Server 7 2004Q2. This application reflects all aspects that comprise a traditional J2EE application.

Use of the Sun Java System Migration Tool is highly recommended for migrating J2EE components without having to manually modify the code. The Migration Tool automates the task of migrating various J2EE Components to Sun Java System Application Server 7 2004Q2.

The iBank application covers the following features of the J2EE specification:

- Servlets, especially with redirection to JSP pages (model-view-controller architecture)
- JSP pages, especially with static and dynamic inclusion of pages
- JSP custom tag libraries
- Creation and management of HTTP sessions
- Database access through the JDBC API

- Enterprise JavaBeans: Stateful and Stateless session beans, CMP and BMP entity beans.
- Assembly and deployment in line with the standard packaging methods of the J2EE application

The iBank application is presented in detail in *Appendix A - iBank Application Specification*.

Migrating J2EE Components

The following migration processes are described in this section:

- [Migrating JDBC Code](#)
- [Migrating Java Server Pages and JSP Custom Tag Libraries](#)
- [Migrating Servlets](#)
- [Obtaining a Data Source from the JNDI Context](#)
- [EJB Migration](#)
- [EJB Changes Specific to Sun Java System Application Server](#)

Migrating JDBC Code

With the JDBC API, there are two methods of database access:

- Establishing Connections Through the DriverManager Interface
(JDBC 1.0 API), by loading a specific driver and providing a connection URL. This method is used by other Application Servers, such as IBM's WebSphere 4.0
- Using JDBC 2.0 Data Sources

The *Data Source* interface (JDBC 2.0 API) can be used via a configurable connection pool. According to J2EE 1.2, a data source is accessed through the JNDI naming service

NOTE Sun Java System Application Server 7 2004Q2 does not support the Native Type 2 JDBC drivers bundled with Sun ONE Application Server 6.x. You must manually migrate code that uses the Type 2 drivers to use Third Party JDBC drivers.

Establishing Connections Through the DriverManager Interface

Although this means of accessing a database is not recommended, as it is obsolete and is not very effective, there may be some applications that still use this approach.

In this case, the access code will be similar to the following:

```
public static final String driver = "oracle.jdbc.driver.OracleDriver";
public static final String url =
    "jdbc:oracle:thin:tmb_user/tmb_user@iben:1521:tmbank";
Class.forName(driver).newInstance();
Properties props = new Properties();
props.setProperty("user", "tmb_user");
props.setProperty("password", "tmb_user");
Connection conn = DriverManager.getConnection(url, props);
```

This code can be fully ported from Sun ONE Application Server 6.0/6.5 to Sun Java System Application Server 7 2004Q2, as long as the Application Server is able to locate the classes needed to load the right JDBC driver. In order to make the required classes accessible to the application deployed in Sun Java System Application Server 7 2004Q2, you should:

- Place the archive (JAR or ZIP) for the driver implementation in the */lib* directory of the Sun Java System Application Server 7 2004Q2 installation directory.

Modify the *CLASSPATH* by setting the path for the driver through the GUI of the admin server. Click the server instance “server1” and then click the tab “JVM Settings” from the right pane. Now click the option Path Settings and add the path in the classpath suffix text entry box. Once you make the changes, click “Save” and then apply the new settings. Restart the server to modify the configuration file, *server.xml*.

Using JDBC 2.0 Data Sources

Using JDBC 2.0 data sources to access a database provides performance advantages such as transparent connection pooling, enhances productivity by simplifying code and implementation, and provides code portability.

Using a data source in an application requires an initial configuration phase followed by a registration of the data source in the JNDI naming context of the Application Server. Once the data source is registered, the application will easily be able to obtain a connection to the database by retrieving the corresponding *DataSource* object from the JNDI context. The actions are described in the following topics:

- [Configuring a Data Source](#)
- [Looking Up the Data Source Via JNDI To Obtain a Connection](#)

Configuring a Data Source

In Sun ONE Application Server 6.0 data sources and their corresponding JDBC drivers are configured from the server's graphic administration console. Connection pools are managed automatically by the application server, and the administration tool can be used to configure their properties. With integrated type 2 JDBC drivers, the connection pooling properties are defined on a per-driver basis, and common to all data sources using a given driver.

On the other hand, for third-party JDBC drivers, connection pool properties are defined on a per-data source basis. Third-party JDBC drivers can be configured either from the administration tool, or from a separate utility (`db_setup.sh` in Sun Solaris, and `jdbcssetup` in Windows NT/2000). Moreover, the command line utility `iasdeploy` can be used to configure a data source from an XML file describing its properties. These utilities are all located in the `/bin/` sub-directory of the Sun Java System Application Server installation root directory.

In Sun Java System Application Server 7 2004Q2, data sources can be configured from the server's graphic administration console or through the command line utility `asadmin`. The command line utility `asadmin` can be invoked by executing `asadmin` file in Solaris, available in Application Server installation's `bin` directory. At the `asadmin` prompt, use the following commands to create connection pool and JNDI resource.

The syntax for calling the *asadmin* utility to create a connection pool is as follows:

```
asadmin>create-jdbc-connection-pool -u username -w password -H hostname
-p adminport [-s] [--instance instancename] --datasourceclassname
classname [--steadypoolsize=8] [--maxpoolsize=32] [--maxwait=60000]
[--poolresize=2] [--idletimeout=300] [--isconnectvalidatereq=false]
[--validationmethod=auto-commit] [--validationtable tablename]
[--failconnection=false] [--description text] [--property
(name=value)[:name=value]*] connectionpoolid
```

For example:

```
asadmin>create-jdbc-connection-pool -u admin -w password -H c11 -p 4848
--instance server1 --datasourceclassname
oracle.jdbc.pool.OracleConnectionPoolDataSource --property
(user-name=ibank_user):(password=ibank_user) oraclepool
```

Here JDBC connection pool 'oraclepool' for oracle database is created using database schema having the username 'ibank_user' and password 'ibank_user'.

The syntax to create a JDBC resource is as follows:

```
asadmin>create-jdbc-resource -u username -w password -H hostname -p
adminport [-s] [--instance instancename] --connectionpoolid id
[--enabled=true] [--description text] [--property
(name=value)[:name=value]*] jndiname
```

For example:

```
asadmin>create-jdbc-resource -u admin -w password -H c11 -p 4848
--instance server1 --connectionpoolid oraclepool jdbc/IBANK
```

Here jdbc resource is created for the connection pool created above with the JNDI name 'jdbc/IBANK'.

Here is the procedure to follow when registering a data source in Sun Java System Application Server 7 2004Q2 through graphical interface.

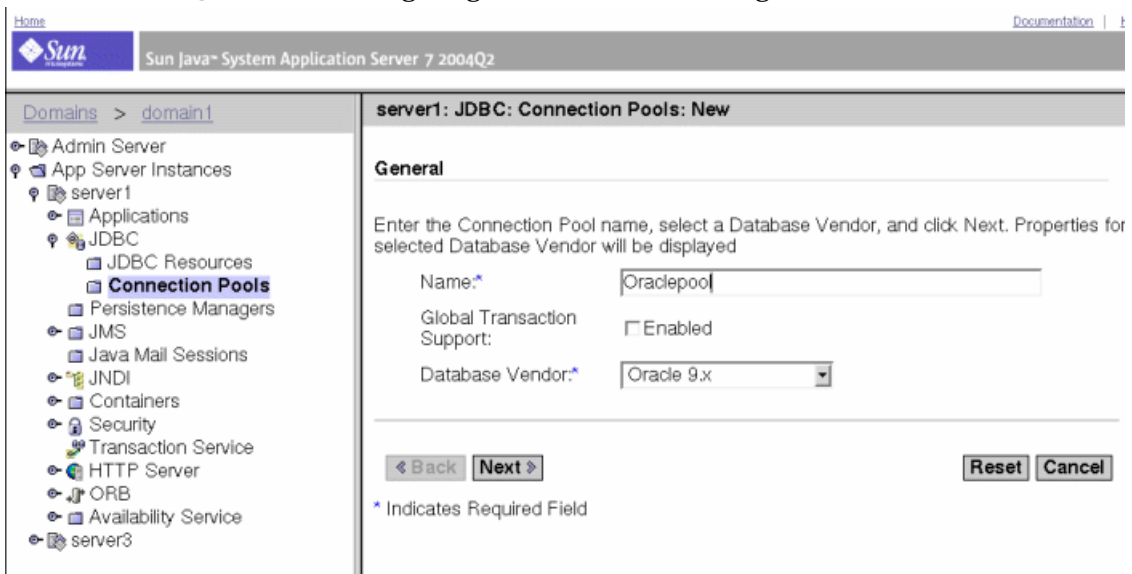
1. Register the data source classname
 - a. Place the archive (JAR or ZIP) for the data source class implementation in the */lib* directory of the Sun Java System Application Server 7 2004Q2 installation directory.
 - b. Modify the CLASSPATH by setting the path for the driver through the GUI of the admin server. Click at the server instance "server1" and then click at tab "JVM Settings", now click at path settings and add the path at the classpath suffix column. Once you make the changes save it and then apply these new settings. Restart the server, which would modify the configuration file, *server.xml*.

2. Register the data source

In Sun Java System Application Server 7 2004Q2, data sources and their corresponding JDBC drivers are configured from the server's graphic administration interface.

The left pane is a tree view of all items you can configure in the Sun Java System Application Server. Click on the item Connection pool at the left pane, the right pane would display the page associated with it where the relevant entries can be made.

Figure 3-2 Configuring Connection Pool through GUI



Similarly now click at the item Data source, right pane would show the entries required for data source setup.

Sun Java System Application Server 7 2004Q2 specific deployment descriptor *sun-web.xml* has to be modified accordingly.

For example if a new data source is configured for the iBank application, the sun-web.xml would have following entries.

```
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application
2.3.1//EN" "Http://localhost:8000/sun-web-app_2_3-1.dtd">

<sun-web-app>

    <resource-ref>

        <res-ref-name>jdbc/iBank</res-ref-name>

        <jndi-name>jdbc/iBank</jndi-name>

        <default-resource-principal>

            <name>ibank_user</name>

            <password>ibank_user</password>

        </default-resource-principal>

    </resource-ref>

</sun-web-app>
```

Looking Up the Data Source Via JNDI To Obtain a Connection

To obtain a connection from a data source, the process is as follows:

- Obtain an initial JNDI context
- Obtain a reference to the data source by using a JNDI lookup
- Obtain a connection using this reference

1. Obtaining the initial JNDI context

To guarantee portability between different environments, the code used to retrieve an InitialContext object (in a servlet, in a JSP page, or an EJB), should be simply, as follows:

```
InitialContext ctx = new InitialContext();
```

2. Obtaining a data source reference

To obtain a reference to a data source bound to the JNDI context, look up the data source's JNDI name from the initial context object. The object retrieved in this way should then be cast as a DataSource type object:

```
ds = (DataSource)ctx.lookup(JndiDataSourceName);
```

3. Obtaining the connection

This operation is very simple, and requires the following line of code:

```
conn = ds.getConnection();
```

Application Server 6.0/6.5, 7, and 7 2004Q2 all follow the above technique for obtaining a connection from data source. So to summarize migration does not require any modification to be made to the code.

Migrating Java Server Pages and JSP Custom Tag Libraries

Sun ONE Application Server 6.0/6.5 complies with the JSP 1.1 specification and Sun Java System Application Server 7 2004Q2 complies with the JSP 1.2 specification.

JSP 1.2 specification contains many new features as well as corrections and clarifications of areas that were not quite right in JSP 1.1 specification.

The most significant changes are:

- JSP 1.2 is based on Servlet 2.3 and Java 2. JSP 1.2 applications will not run on platforms that only support JDK 1.1. JSP 1.2 is backward compatible with JSP 1.1, so JSP 1.1 application should run without any tweaking in a JSP 1.2 complaint container.
- The definition of XML syntax for a JSP page has been finalized. So a JSP 1.2 complaint container must accept files in both JSP 1.1 format and the new XML format called as JSP document.
- Tag libraries can make use of Servlet 2.3 event listeners.
- A new type of validation has been added, for the tag libraries, which validates JSP pages.
- New options for tag library distribution and deployment have been added.

These changes are basically enhancements and are not required to be made, while migrating JSP pages from JSP API 1.1 to 1.2.

Migrating JSP Custom Tag Libraries

When you are attempting to migrate applications that make use of custom tag library under Sun Java System Application Server 7 2004Q2 platform, following approaches can be envisaged:

The simplest approach is to try using the 6.x application custom tag library under Sun Java System Application Server 7 2004Q2 platform. There are several caveats to this:

- First approach is to try redeveloping a custom tag library that would emulate the functionality provided by the server custom tags. Due to sheer amount of work that this approach entails, it is not recommended unless an application has huge dependency on the tag library.
- Second approach is to rewrite pages in the application so that they may no longer use any functionality of the proprietary tag library. Obviously, this approach can involve a substantial investment, but is likely to be the most cost-effective solution for applications that have critical dependencies on the custom tag library.
- Third approach would be to use JavaServer Pages Standard Tag Library (JSTL) equivalent for the custom tag libraries. JSTL provides a framework for integrating existing custom tags with JSTL tags. However, note that this approach may not give the exact functionality.
- The easiest approach is to use the Migration Tool. Use of this tool will eliminate the effort of manual migration of custom tag libraries and takes care of the migration. See [Appendix B, “Migration Resources,”](#) for more information on the Migration Tool.

Migrating Servlets

Sun ONE Application Server 6.0 and 6.5 support the Servlet 2.2 API whereas Sun Java System Application Server 7 2004Q2, supports the Servlet 2.3 API.

Servlet API 2.3 actually leaves the core of servlets relatively untouched; most changes are concerned with adding new features outside the core.

The most significant features are:

- Servlets now require JDK 1.2 or later
- A filter mechanism has been created
- Application lifecycle events have been added
- New internationalization support has been added
- New error and security attributes have been added
- The `HttpUtils` class has been deprecated

- Several DTD behaviors have been expanded and clarified

These changes are basically enhancements and are not required to be made while migrating servlets from Servlet API 2.2 to 2.3.

However, if the servlets in the application use JNDI to access resources of the J2EE application (such as data sources, EJBs, and so forth), some modifications may be needed in the source files or in the deployment descriptor.

These modifications are explained in detail in the following sections:

- [Obtaining a Data Source from the JNDI Context](#)
- [Declaring EJBs in the JNDI Context](#)

Obtaining a Data Source from the JNDI Context

To obtain a reference to a data source bound to the JNDI context, look up the data source's JNDI name from the initial context object. The object retrieved in this way should then be *cast* as a `DataSource` type object:

```
ds = (DataSource)ctx.lookup(JndiDataSourceName);
```

For detailed information, refer to section “Migrating JDBC Code” in the previous pages.

Declaring EJBs in the JNDI Context

Refer to section [Declaring EJBs in the JNDI Context](#) from [Appendix C, “Migrating from the Enterprise Java Beans 1.1 Specification to Enterprise Java Beans 2.0.”](#)

EJB Migration

As mentioned in [Chapter 2, “Migrating to Sun Java System Application Server, Overview,”](#), while Sun ONE Application Server 6.0 and 6.5 support the EJB 1.1 specification, Sun Java System Application Server 7 2004Q2 also supports the EJB 2.0 specification. The EJB 2.0 specification introduces the following new features and functions to the architecture:

- Message Driven Beans (MDBs)
- Improvements in Container-Managed Persistence (CMP)
- Container-managed relationships for entity beans with CMP
- Local interfaces
- EJB Query Language (EJB QL)

Although the EJB 1.1 specification will continue to be supported in Sun Java System Application Server 7 2004Q2, the use of the EJB 2.0 architecture is recommended to leverage its enhanced capabilities.

For detailed information on migrating from EJB 1.1 to EJB 2.0, refer to [Appendix C, “Migrating from the Enterprise Java Beans 1.1 Specification to Enterprise Java Beans 2.0.”](#)

EJB Changes Specific to Sun Java System Application Server

Migrating EJB's from Sun ONE Application server 6.0/6.5 to Sun Java System Application Server 7 2004Q2 would not require any changes in the EJB code. The following DTD changes are required.

Session Beans

- The `<!DOCTYPE>` definition should be modified to point to the latest DTDs with J2EE standard DDs, such as `ejb-jar.xml`.
- Replace `ias-ejb-jar.xml` file with the modified version of this file, named `sun-ejb-jar.xml`, created manually according to the DDs.
- In `sun-ejb-jar.xml`, the JNDI name for all the EJB's should prepend 'ejb/' in all the JNDI names. This is required as, in Sun ONE Application Server 6.5, the JNDI name of the EJB could only be `ejb/<ejb-name>` where `<ejb-name>` is the name of the EJB as declared inside `ejb-jar.xml`.

In Sun Java System Application Server 7 2004Q2, a new tag in the `sun-ejb-jar.xml`, is used to declare the JNDI name of the EJB.

NOTE To avoid changing JNDI names throughout the application, we recommend that the JNDI name of the EJB should be declared as `ejb/<ejb-name>` inside the `<jndi-name>` tag.

Migrating EJB Applications that Support SFSB Failover (Enterprise Edition)

Sun ONE Application Server 6.5 supports failover of stateful session beans. To take advantage of the SFSB failover in 6.5, the session bean need to be configured with failover and DSync. The DSync (Distributed Store) mechanism is used to save the session beans's conversational state during runtime.

NOTE Sun ONE Application Server 6.5 does not support failover of stateful session beans for rich clients on the RMI/IIOP path. Such applications can take advantage of SFSB failover on the RMI/IIOP path in Sun Java System Application Server 7 2004Q2. For more information on SFSB failover configuration, see the "Session Persistence" chapter of the *Sun Java System Application Server Administration Guide*.

Sun Java System Application Server 7 2004Q2, Enterprise Edition supports failover of stateful session beans. Application Server 7 2004Q2 uses the High Availability DataBase (HADB) for storing session data. The principle followed in supporting SFSB failover in saving the conversational state of an SFSB at pre-defined points in its lifecycle to a persistent store. This mechanism is referred to as 'checkpointing'. In case of a server crash, the checkpointed state of an SFSB can be retrieved from the persistent store. In order to use HADB for storing session data, you must configure HADB as the persistent store. The underlying store for the HTTP sessions and stateful session beans is same and the configuration of persistent store is exactly similar to configuration of session store.

For information on configuring HADB for session store, see the *Sun Java System Application Server Administration Guide*.

Migration of stateful session beans deployed in Sun ONE Application Server 6.5 to Sun Java System Application Server 7 2004Q2 does not require any changes in the EJB code. However, the following steps must be performed:

- The `<!DOCTYPE` definition should be modified to point to the latest DTD url in case of J2EE standard DDs, like `ejb-jar.xml`.
- Replace `ias-ejb-jar.xml` with the modified version of this file, i.e., `sun-ejb-jar.xml`, which is created manually according to the DTDs.
- Replace all the `<ejb-name>-ias-cmp.xml` files with one `sun-cmp-mappings.xml` file, which is created manually.
- No changes are required in the application source code for taking advantage of the SFSB state failover support. All configuration needed for checkpointing SFSBs will be applied at the Application Server specific deployment descriptor (`sun-ejb-jar.xml`), or in the server configuration file (`server.xml`).

For more information see the *Sun Java System Application Server Administration Guide*.

However, if you are accessing the ejb's through webpath(i.e., servlets) then you need to store the ejb home and remote references in the session. The following is the code example to store ejbHome and ejbRemote interfaces in the session:

```
session.setAttribute("ejbhome", ejbHome);
session.setAttribute("ejbremote", ejbRemote);
```

The following code example demonstrates how to retrieve the ejbHome and ejbRemote from the session:

```
ejbHome = session.getAttribute("ejbhome");
ejbRemote = session.getAttribute("ejbremote");
```

- In the server.xml, make sure that the availability-enabled attribute of availability-service element is set to TRUE. If availability-enabled attribute is set to TRUE indicates that failover is enabled at the server instance level. That is, if a server instance fails to process a request, the request is routed to the next available server instance.

SFSB checkpointing adds performance overhead on the EJB container, you may want to restrict checkpointing to a list of SFSBs whose state failover is critical to the application.

You can enable/disable the checkpointing at the method level. This is configurable in sun-ejb-jar.xml. For more details see the “Enabling SFSB Checkpointing” section of the *Sun Java System Application Server Developer's Guide to Enterprise JavaBeans Technology*.

If, in the deployment descriptor for the SFSB ejb module in 6.5 (ias-ejb-jar.xml), the failoverrequired attribute of the session element is set to TRUE, you might want to enable availability-service for such ejb modules in the Application Server 7 2004Q2 environment.

Entity Beans

- The <!DOCTYPE> definition should be modified to point to the latest DTDs with J2EE standard DDs, such as ejb-jar.xml.
- Insert <cmp-version> tag with the value 1.1 for all CMPs in ejb-jar.xml.
- Generate dbschema by using the capture-/spaceschema utility in the Sun Java System Application Server 7 2004Q2 installation's bin directory and place it above META-INF folder for Entity beans.
- ias-ejb-jar.xml should be replaced with its new version, named sun-ejb.jar.xml, in Sun Java System Application Server 7 2004Q2.

- In Sun ONE Application Server 6.5, the finders sql was directly embedded inside the `<ejb-name>-ias-cmp.xml`. In Sun Java System Application Server, this has changed such that, now mathematical expressions are used to declare the `<query-filter>` for the various finder methods.

Message Driven Beans

Sun Java System Application Server 7 2004Q2 provides seamless Message Driven Support through the tight integration of Sun Java System Message Queue 3.5 with the Application Server, providing a native, built-in JMS Service.

This installation provides Sun Java System Application Server with a JMS messaging system that supports any number of Application Server instances. Each server instance, by default, has an associated built-in JMS Service that supports all JMS clients running in the instance.

Both container-managed and bean-managed transactions as defined in the Enterprise JavaBeans Specification, v2.0 are supported.

Message Driven Bean support in Sun ONE Application Server 6.x was restricted to developers, and used many of the older proprietary APIs. Messaging services were provided by Sun ONE Message Queue for Java 2.0. An LDAP directory was also required under Sun ONE Application Server 6.x to configure the `QueueConnectionFactory` object.

The `QueueConnectionFactory`, and other particulars required to configure Message Driven Beans in Sun Java System Application Server should be specified in the `ejb-jar.xml` file.

For more information on the changes to deployment descriptors, see [“Migrating Deployment Descriptors.”](#) For information on Message Driven Bean implementation in Sun Java System Appserver 7 2004Q2, see *Sun Java System Application Server Developer’s Guide to Enterprise Java Bean Technology*.

Migrating Web Applications

Sun ONE Application Server 6.0 and 6.5 support servlets (Servlet API 2.2), and JSPs (JSP 1.1). Sun Java System Application Server 7 2004Q2 on the other hand supports servlets (Servlet API 2.3) and JSPs (JSP 1.2).

Within these environments it is essential to group the different components of an application (servlets, JSP and HTML pages and other resources) together within an archive file (J2EE-standard Web application module) before you can deploy it on the application server.

According to the J2EE 1.3 specification, a Web application is an archive file (.WAR file) with the following structure:

- a root directory containing the HTML pages, JSP, images and other "static" resources of the application.
- a *META-INF/* directory containing the archive manifest file (MANIFEST.MF) containing the version information for the SDK used and, optionally, a list of the files contained in the archive.
- a *WEB-INF/* directory containing the application deployment descriptor (*web.xml* file) and all the Java classes and libraries used by the application, organized as follows:
 - a *classes/* sub-directory containing the tree-structure of the compiled classes of the application (servlets, auxiliary classes), organized into packages.
 - a *lib/* directory containing any Java libraries (.jar files) used by the application.

Migrating Web Application Modules

Migrating applications from Sun ONE Application server 6.0/6.5 to Sun Java System Application Server 7 2004Q2 would not require any changes in the Java/JSP code. The following changes are, however, still required.

- *web.xml*

Sun Java System Application Server 7 2004Q2 adheres to J2EE 1.3 standards, according to which, the *web.xml* file inside a WAR should adhere to the revised DTD available at http://java.sun.com/dtd/web-app_2_3-1.dtd. This DTD fortunately, is a superset of the previous versions' DTD, hence only the `<!DOCTYPE` definition needs to be changed inside the *web.xml*, which is to be migrated. The modified `<!DOCTYPE` declaration should look like:

```
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web
Application 2.3//EN" "http://java.sun.com/dtd/web-app_2_3-1.dtd">
```

- *ias-web.xml*

In Sun Java System Application Server 7 2004Q2 the name of this file is changed to *sun-web.xml*.

This XML file is required to declare the Sun Java System Application Server 7 2004Q2 specific properties/resources that will be required by the web application.

Note: See the next section for some important inclusions to this file.

If the `ias-web.xml` of the Sun ONE Application Server 6.5 application is present and does declare Sun ONE Application Server 6.5 specific properties, then this file needs to be migrated to Sun Java System Application Server 7 2004Q2 standards. The DTD file name has to be changed to `sun-web.xml`.

Once the `web.xml` and `ias-web.xml` are migrated in the above-mentioned fashion, the Web application (.WAR archive) can be deployed from the Sun Java System Application Server 7 2004Q2's graphical user interface of the admin server or from the command line utility ***asadmin***, where the deployment command should mention the ***type of application as web***.

The command line utility `asadmin` can be invoked by running `asadmin.bat` file kept at Sun Java System Application Server 7 2004Q2 installation's bin directory.

The command at *asadmin* prompt would be:

```
asadmin> deploy -u username -w password -H hostname -p adminport --type
web [--contextroot contextroot] [--force=true] [--name component-name]
[--upload=true] [--instance instancename] filepath
```

Deployment can also be done from the Sun ONE Studio for Java development environment. For more information, see the Sun ONE Studio for Java documentation.

Particular setbacks when migrating servlets and JSPs

The actual migration of the components of a Servlet / JSP application from Sun ONE Application Server 6.0/6.5 to Sun Java System Application Server 7 2004Q2 will not require any modifications to be made to the component code.

In case if the web-application is using a server resource, for example, a `DataSource`, then Sun Java System Application Server 7 2004Q2 requires that this resource be declared inside the `web.xml` and correspondingly inside `sun-web.xml`. For declaring a `DataSource` called `jdbc/iBank`, the `<resource-ref>` tag as declared inside the `web.xml` would look like this:

```
<resource-ref>
    <res-ref-name>jdbc/iBank</res-ref-name>
    <res-type>javax.sql.XADataSource</res-type>
    <res-auth>Container</res-auth>
    <res-sharing-scope>Shareable</res-sharing-scope>
</resource-ref>
```

Corresponding declaration inside the `sun-web.xml` will look like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<! DOCTYPE FIX ME: need confirmation on the DTD to be used for this file
<sun-web-app>
    <resource-ref>
        <res-ref-name>jdbc/iBank</res-ref-name>
        <jndi-name>jdbc/iBank</jndi-name>
    </resource-ref>
</sun-web-app>
```

Migrating Enterprise EJB Modules

Sun ONE Application Server 6.0 and 6.5 support the EJB 1.1 API whereas Sun Java System Application Server 7 2004Q2 supports the EJB 2.0 API. Thereby, both can support:

- Stateful or Stateless Session Beans.
- Entity beans with bean managed persistence (BMP), or container managed persistence (CMP).

EJB 2.0 API however, introduces a new type of enterprise bean, called a message-driven bean in addition to the session and entity beans.

J2EE 1.3 specification dictates that the different components of an EJB must be grouped together in a JAR file with the following structure:

- `META-INF/` directory with an XML deployment descriptor named *`ejb-jar.xml`*
- The `.class` files corresponding to the home interface, remote interface, the implementation class, and the auxiliary classes of the bean with their package.

Sun Java System Application Servers observe this archive structure. However, the EJB 1.1 specification leaves each EJB container vendor to implement certain aspects as they see fit:

- Database persistence of CMP EJBs (particularly the configuration of mapping between the bean's CMP fields and columns in a database table).
- Implementation of the custom finder method logic for CMP beans.

As we might expect, Sun ONE Application Server 6.0 or 6.5 and 7 and Sun Java System Application Server 7 2004Q2 diverge on certain points, which means that when migrating an application certain aspects require particular attention. Some XML files have to be modified:

- The `<!DOCTYPE` definition should be modified to point to the latest DTD url in case of J2EE standard DDs, like `ejb-jar.xml`.
- Replace `ias-ejb-jar.xml` with the modified version of this file, i.e., `sun-ejb-jar.xml`, which is created manually according to the DTDs.
- Replace all the `<ejb-name>-ias-cmp.xml` files with one `sun-cmp-mappings.xml` file, which is created manually.
- Only for CMP entity beans: Generate `dbschema` by using the *capture-schema* utility in the Sun Java System Application Server 7 2004Q2 installation's `bin` directory and place it above `META-INF` folder for the Entity bean.

Migrating Applications to Support HTTP Failover (Enterprise Edition)

Sun ONE Application Server 7 and Sun Java System Application Server, Enterprise Edition 7 2004Q2 supports load balancing and HTTP session persistence. The primary goal of load-balancing is to distribute the work load between multiple server instances, thereby increasing overall throughput of the system.

The Application Server uses sticky round-robin algorithm to load balance the incoming HTTP or HTTPS requests.

For information on configuring HTTP failover, see the *Sun Java System Application Server Administration Guide*.

To migrate 6.x HTTP applications to 7 2004Q2 environment and enable load-balancing capabilities, perform the following steps. Note that, no code changes will be required in the application.

1. Make sure that at least two application server instances are created and configured.
2. Rename the `ias-web-app.xml` to `sun-web.xml`. For more information on migrating the deployment descriptors, see the [“Migrating Deployment Descriptors” on page 67](#).

3. Update the <DOCTYPE definition with the following code:

```
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web
Application 2.3//EN"
'http://java.sun.com/j2ee/dtds/web-app_2_3-1.dtd'>
```

4. In Sun ONE Application Server 6.5, the failover of HTTP applications was based on Dsync mechanism. The configuration for HTTP failover was done in the `ias-web-app.xml` file.

The `<server-info>` element defined under the `<servlet-info>` element, specifies whether the server on which the servlet will be served from is enabled.

The `<session-info>` element defines the following:

- `dsync-type`: This can take the value `dsync-distributed` or `dsync-local`.
`dsync-distributed` implies that the session is distributed and thus available on all configured servers.
`dsync-local` implies that the session is available only on the server on which the session was created.
- `impl`: This can take the values `distributed` or `lite`.
`distributed` implies that the session is distributed.
`lite` implies that the session is local to the Java engine where the session was created. If this value is set, the `dsync-type` setting is ignored. I

In Sun Java System Application Server 7 2004Q2, to enable failover of applications on the HTTP route, you define the following properties in the sun-specific web application deployment descriptor file: `sun-web.xml`.

- `persistence-store` - This can take the values `memory`, `file`, or `ha`. In 6.5, however, only memory based persistence store was supported.
- `persistence-scope` - define the scope of persistence.
 - `session` - For every session, the session information will be saved.
 - `modified-session` - Only the modified session data will be stored.
 - `modified-attribute` - Only the modified attribute data will be stored. In 6.5, only `modified-attribute` scope was supported.
- `persistenceFrequency` - The frequency can be for every web method or time based. In 6.5, only web-method was supported.

- **web-method** - The session state is stored at the end of each web request prior to sending a response back to the client. This mode provides the best guarantee that the session state is fully updated in case of failure.
- **time-based** - The session state is stored in the background at the specified frequency. This mode provides less of a guarantee that the session state is fully updated. However, it can provide a significant performance improvement because the state is not stored after each request.

A sample of the `sun-web.xml` file is given below:

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE sun-web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Sun ONE
Application Server 7.1 Servlet 2.3//EN"
'http://www.sun.com/software/sunone/appserver/dtds/sun-web-app_2_3-
1.dtd'>

<sun-web-app>

  <session-config>

    <session-manager>

      <manager-properties>

        <property = persistence-type, value = "ha">
        <property="persistenceFrequency, value = "web-based">

      </manager-properties>

      <store-properties>

        <property = "persistenceScope" value="session">

      </store-properties>

    </session-manager>

  </session-config>

</sun-web-app>
```

For more information on `sun-web.xml` configuration file, see the *Sun Java System Application Server Developer's Guide to Web Applications*.

5. Sun Java System Application Server 7 2004Q2 requires the load balancer plug-in to be installed and configured, in order to load-balance the HTTP request and failover the requests to available server instances in a cluster when there is a failure.

For more information about the load balancer, see the *Sun Java System Application Server Administration Guide*.

6. In the `load-balancer.xml` file, make sure that the `web-module enabled` element is set to `true`.

```
<loadbalancer>

<cluster name=cluster1>

...

<web-module context-root="abc" enabled=true>

</cluster>

<property name="https-routing" value="true"/>

</loadbalancer>
```

`enabled=true` specifies that the web module is active (enabled) for requests to be load balanced to it.

7. Define the `https-routing` property and set its value to `true`.

For more information on editing the `load-balancer.xml` file, see the *Sun Java System Application Server Administration Guide*.

Deploy the applications on all server instances that is participating in load balancing.

Migrating Enterprise Applications

According to the J2EE specifications, an enterprise application is an EAR file, which must have the following structure:

- a `META-INF/` directory containing the XML deployment descriptor of the J2EE application called *application.xml*.
- the `.JAR` and `.WAR` archive files for the EJB modules and Web module of the enterprise application, respectively.

In the application deployment descriptor, we define the modules that make up the enterprise application, and the Web application's context root.

Sun ONE Application server 6.0/6.5 and 7 primarily supports the J2EE model wherein applications are packaged in the form of an enterprise archive (EAR) file (extension .ear). The application is further subdivided into a collection of J2EE modules, packaged into Java archives (JAR, extension .jar) for EJBs and web archives (WAR, extension .war) for servlets and JSPs.

It is therefore essential to follow the steps listed here before deploying an enterprise application:

- Package EJBs in one or more EJB modules,
- Package the components of the Web application in a Web module,
- Assemble the EJB modules and Web modules in an enterprise application module,
- Define the name of the enterprise application's root context, which will determine the URL for accessing the application.

NOTE Sun Java System Application Server 7 2004Q2 uses a new class loader hierarchy as compared to Sun ONE Application Server 6.0/6.5. In the new scheme, for a given application, one class loader loads all EJB modules and another class loader loads web modules. These two are related in a parent child hierarchy where the JAR module class loader is the parent module of the WAR module class loader. Hence all classes loaded by the JAR class loader are available/ accessible to the WAR module but the reverse is not true. Suppose there is a certain class which is required by the JAR as well as the WAR, then it should be packaged inside the JAR module only. If this guideline is not followed it would lead to class conflicts.

Application Root Context and Access URL

There is one particular difference between Sun ONE Application Server 6.0/6.5 and Sun Java System Application Server 7 2004Q2, concerning the applications access URL (root context of the application's Web module):

If `AppName` is the name of the root context of an application deployed on a server called `hostname`, then the access URL for this application will differ depending on the application server used:

- With Sun ONE Application Server 6.0 or 6.5, which is always used jointly with a Web front-end, the access URL for the application will take the following form (assuming the Web server is configured on the standard HTTP port, 80):

`http://hostname/NASApp/AppName/`

- With Sun ONE Application Server 7 and Sun Java System Application Server 7 2004Q2, the URL will take the form:

`http://hostname:port/AppName/`

The TCP port used as default by Sun ONE Application Server 7 is port 80.

Although the difference in access URLs between Sun ONE Application Server 6.0/6.5 and Sun Java System Application Server 7 2004Q2 may appear minor, it can however be problematical when migrating applications that make use of absolute URL references. In such cases, it will be necessary to edit the code to update any absolute URL references so that they are no longer prefixed with the specific marker used by the Web Server plug-in for Sun ONE Application Server 6.0/6.5.

If you are using the Migration Tool, it automatically searches for any occurrences of `/NASApp` in the input application (only if the application is a Sun ONE Application Server 6.5 application), and replaces it with `''`. The migration report lists down all occurrences of such instances, so that you are aware of this change. If you are migrating a rich client application that contains hard-coded urls with `/NASApp`, consider any of the following workarounds:

- Modify the application context root to change from `'app'` to `'/NASApp/app'`. While this will help the Application Server 7 2004Q2 plug-in forward the request to the correct application it would fail in places where an application performs operations based on application context value.
- Modify client application to remove any references to `'NASApp'`.

Applications With Form-based Authentication

Applications developed on Sun ONE Application Server 6.5 that use form-based authentication can pass the request parameters to the Authentication Form or the Login page. The Login page could be customized to display the authentication parameters based on the input parameters.

For example:

`http://gatekeeper.uk.sun.com:8690/NASApp/test/secured/page.jsp?arg1=test&arg2=m`

Sun Java System Application Server 7 2004Q2 does not support the passing of request parameters while displaying the Login page. The applications that uses form-based authentication, which passes the request parameters can not be migrated to Sun Java System Application Server 7 2004Q2. Porting such applications to Application Server 7 2004Q2 requires significant changes in the code. Instead, you can store the request parameter information in the session, which can be retrieved while displaying the Login page.

The following code example demonstrates the workaround:

Before changing the code in 6.5:

```
-----index-65.jsp -----

<%@page contentType="text/html"%>
<html>

    <head><title>JSP Page</title></head>
    <body>

        go to the <a href="secured/page.htm">secured area</a>

    </body>

</html>

-----login-65.jsp-----

<%@page contentType="text/html"%>
<html>
<head> </head>
<body>

    <!-- Print login form -->

    <h3>Parameters</h3><br>
    out.println("arg1 is " + request.getParameter("arg1"));
    out.println("arg2 is " + request.getParameter("arg2"));

</body>
</html>
```

After changing the code in 7 2004Q2:

```
-----index-7.jsp -----

<%@page contentType="text/html"%>
<html>

    <head><title>JSP Page</title></head>
    <body>
```

```

<%session.setAttribute("arg1","test"); %>
<%session.setAttribute("arg2","me"); %>
go to the <a href="secured/page.htm">secured area</a>

</body>

</html>

```

The `index-7.jsp` shows how you can store the request parameters in a session.

```

-----login-7.jsp-----

<%@page contentType="text/html"%>
<html>
<head> </head>
<body>

    <!-- Print login form -->
    <h3>Parameters</h3><br>

    <!--retrieving the parameters from the session -->
    out.println("arg1 is" + (String)session.getAttribute("arg1"));
    out.println("arg2 is" + (String)session.getAttribute("arg2"));

</body>
</html>

```

Migrating Proprietary Extensions

A number of classes proprietary to the Sun ONE Application Server 6.0/ 6.5 environment may have been used in applications. Some of the proprietary Sun ONE packages used by Sun ONE Application Server 6.x are listed below:

- `com.iplanet.server.servlet.extension`
- `com.kivasoft.dlm`
- `com.iplanetiplanet.server.jdbc`
- `com.kivasoft.util`
- `com.netscape.server.servlet.extension`
- `com.kivasoft`
- `com.netscape.server`

These APIs are not supported in Sun Java System Application Server 7 2004Q2. Applications using any classes belonging to the above package will have to be rewritten such that the applications use standard J2EE APIs. Applications using Custom JSP tags and UIF framework also needs to be rewritten to use standard J2EE API.

Migrating UIF

Sun Java System Application Server 7 2004Q2 does not support the use of Unified Integration Framework (UIF) API for applications. Instead, it supports the use of J2EE Connector Adapter (JCA) for integrating the applications. However, the applications developed in Sun ONE Application Server 6.5 uses the UIF. In order to deploy such applications to Sun Java System Application Server 7 2004Q2, you need to migrate the UIF to JCA. This section discusses the pre-requisites and steps to migrate the applications using UIF to Sun Java System Application Server 7 2004Q2.

Before migrating the applications, you need to make sure that the UIF is installed on Sun ONE Application Server 6.5. To check for the installation, you can use any of the following two approaches:

Approach 1: Checking in the registry files

UIF is installed as a set of application server extensions. They are registered in app server registry during the installation. Search for the following strings in the registry to check whether UIF is installed.

Extension Name Set:

- Extension DataObjectExt-cDataObject
- Extension RepositoryExt-cLDAPRepository
- Extension MetadataService-cMetadataService
- Extension RepoValidator-cRepoValidator
- Extension BSPRuntime-cBSPRuntime
- Extension BSPErrorLogExt-cErrorLogMgr
- Extension BSPUserMap-cBSPUserMap

The registry file on Solaris Operating Environment can be found at the following location:

Install_dir/AS/registry/reg.dat

Approach 2: Checking for UIF binaries in installation directories

UIF installers copy specific binary files in to the application server installation. A successful find of these files below indicate that UIF is installed.

The location of the following files on Solaris and Windows is:

Install_dir/AS/APPS/bin

List of files to be searched on Solaris:

- libcBSPRlop.so
- libcBSPRuntime.so
- libcBSPUserMap.so
- libcDataObject.so
- libcErrorLogMgr.so
- libcLDAPRepository.so
- libcMetadataService.so
- libcRepoValidator.so
- libjx2cBSPRuntime.so
- libjx2cDataObject.so
- libjx2cLDAPRepository.so
- libjx2cMetadataService.so

List of files to be searched on Windows:

- cBSPRlop.dll
- cBSPRuntime.dll
- cBSPUserMap.dll
- cDataObject.dll
- ErrorLogMgr.dll
- cLDAPRepository.dll

- `cMetadataService.dll`
- `cRepoValidator.dll`
- `jx2cBSPRuntime.dll`
- `jx2cDataObject.dll`
- `jx2cLDAPRepository.dll`
- `jx2cMetadataService.dll`

Before migrating the UIF to Sun Java System Application Server 7 2004Q2, make sure that the UIF API is being used in applications. To verify its usage:

- Check for the usage of `netscape.bsp` package name in the java sources.
- Check for the usage of `access_cBSPRuntime.getBSPRuntime` method in the sources. Calling this method is essential in acquiring the UIF runtime.

Migration Process

To migrate the UIF, you can use the Sun ONE Connector Builder tool. This tool is also integrated into Sun ONE Studio and hence you can use Sun ONE Studio to migrate the UIF based applications to JCA based applications. The key features of the Sun ONE Connector Builder Tool are:

- Connector code generation
- Testing and debugging support
- Deployment support
- Creates connector deployment descriptors
- Assembles connectors for deployment
- Administration hooks
- Monitoring hooks
- Customize-able environment

Note: Migration of applications using the UIF in Sun ONE App Sever 6.5 does not require any changes in the code.

For more information on using the Sun ONE Connector Builder tool, visit the following URL:

<http://docs.sun.com/db/coll/sl.conbldr>

Migrating Rich Clients

This section describes the steps for migrating RMI/IIOP and ACC clients developed in Sun ONE Application Server 6.x to Sun Java System Application Server 7 2004Q2 EE.

Authenticating a Client in 6.x

Sun ONE Application Server provides a client-side callback mechanism that enables applications to collect authentication data from the user such as the username and the password. The authentication data collected by the Sun ONE CORBA infrastructure is propagated to the Application Server via IIOP.

If ORBIX 2000 is the ORB used for RMI/IIOP, portable interceptors implement security by providing hooks, or interception points, which define stages within the request and reply sequence.

Authenticating a Client in 7 2004Q2 SE/EE

The authentication is done based on JAAS (Java Authorization and Authentication System API) and the client can that implement a `CallbackHandler`. If a client does not provide a `CallbackHandler`, then the Default `CallbackHandler` called the `LoginModule` will be used by the ACC for obtaining the authentication data.

For detailed instructions on using JAAS for authentication, see the *Sun Java System Application Server Developer's Guide to Clients*.

NOTE Sun Java System Application Server 7 2004Q2 does not support authentication of stand-alone (non-ACC) clients.

Using ACC in 6.x and 7 2004Q2 EE

In 6.x, no separate `appclient` script is provided. You are required to place the `iasacc.jar` file in the classpath instead of the `iascleint.jar` file. The only benefit of using the ACC for packaging application clients in 6.x is that the JNDI names specified in the client application are indirectly mapped to the absolute JNDI names of the EJBs.

In case of 6.x applications, a stand-alone client would use the absolute name of the EJB in the JNDI lookup. That is, outside an ACC, the following approach would be used to lookup the JNDI:

```
initial.lookup("ejb/ejb-name");
initial.lookup("ejb/module-name/ejb-name");
```

If your application was developed using 6.5 SP3, you would have used the prefix “`java:comp/env/ejb/`” when performing lookups via absolute references.

```
initial.lookup("java:comp/env/ejb/ejb-name");
```

In Sun Java System Application Server 7 2004Q2, the JNDI lookup is done on the `jndi-name` of the EJB. The absolute name of the `ejb` must not be used. Also, the prefix, `java:comp/env/ejb` is not supported in Version 7 and 7 2004Q2. Replace the following jar files in the classpath with `appserv-ext.jar`:

```
iasclient.jar, iasacc.jar, or javax.jar
```

Load-balancing and Failover Features in ACC Clients (Enterprise Edition)

Load balancing is handled implicitly by the CXS engine in SunONE Application Server 6.5 upon number of Java engines registered. In Sun Java System Application Server 7 2004Q2, Enterprise Edition, this feature requires explicit configuration details from the clients.

After migrating the deployment descriptors from 6.x to 7 2004Q2, provide the configuration details in the `sun-acc.xml` file to enable failover capabilities in your ACC client. See [“Migrating Deployment Descriptors” on page 67](#) for information on migrating deployment descriptors.

Define the load balancing properties in the `sun-acc.xml` file to provide a highly available ACC client. The properties are defined as property elements in the `sun-acc.xml` file.

- `com.sun.appserv.iiop.endpoints`

This property defines the list of one or more IIOP endpoints. An endpoint is specified as `host:port` where `host` is the name or IP address of the system where Sun Java System Application Server is running. Port is the IIOP port at which the server is listening for IIOP requests.

- `com.sun.appserv.iiop.loadbalancingpolicy`

If the endpoint property is specified, then, this property is used to specify the load balancing policy. The value for this property must be `InitialContext-based`.

For example:

```
<client-container>
  <target-server name="gasol-e1" address="gasol-e1" port="3700">
```

```

        <property name="com.sun.appserv.iiop.loadbalancingpolicy"
value="ic-based" />

        <property name="com.sun.appserv.iiop.endpoints"
value="qasol-e1:3700", "qasol-e1:3800"/>

</client-container>

```

To failover an ACC client on the RMI/IIOP path, information about all the endpoints in a cluster to which the RMI/ IIOP requests can be failed over must be available. You must have defined the IIOP endpoints in the `server.xml` file. The `iiop-cluster` element under the `availability-service` element defines the IIOP endpoints.

For more information, see the *Sun Java System Application Server Administration Guide*.

Migrating Deployment Descriptors

The following table summarizes the deployment descriptor migration mapping.

Source Deployment Descriptor	Target Deployment Descriptor
<code>ejb-jar.xml</code> - 1.1	<code>ejb-jar.xml</code> - 2.0
<code>ias-ejb-jar.xml</code>	<code>sun-ejb-jar.xml</code>
<code><bean-name>-ias-cmp.xml</code>	<code>sun-cmp-mappings.xml</code>
<code>web.xml</code>	<code>web.xml</code>
<code>ias-web.xml</code>	<code>sun-web.xml</code>
<code>application.xml</code>	<code>application.xml</code>

The J2EE standard deployment descriptors `ejb-jar.xml`, `web.xml` and `application.xml` are not modified significantly. However, the `ejb-jar.xml` deployment descriptor is modified to make it compliant with EJB 2.0 specification in order to make the application deployable on Sun Java System Application Server 7 2004Q2.

Majority of the information required for creating `sun-ejb-jar.xml` and `sun-web.xml` comes from `ias-ejb-jar.xml` and `ias-web.xml` respectively. However, there is some information that is required and extracted from the home interface (java file) of the CMP entity bean, in case the `sun-ejb-jar.xml` being migrated declares one. This is

required to build the `<query-filter>` construct inside the `sun-ejb-jar.xml`, which requires information from inside the home interface of that CMP entity bean. If this source file is not present during the migration time, the `<query-filter>` construct will get created, but with lots of missing information (which will manifest itself in the form of "REPLACE ME" phrases in the migrated `sun-ejb-jar.xml`).

Additionally, if the `ias-ejb-jar.xml` contains a `<message-driven>` element, then information from inside this element is picked up and used to fill up information inside both `ejb-jar.xml` and `sun-ejb-jar.xml`. Also, inside the `<message-driven>` element of `ias-ejb-jar.xml`, there is an element `<destination-name>`, which holds the JNDI name of the topic or queue to which the MDB should listen to. In Sun ONE Application Server 6.5, the naming convention for this jndi name is `"cn=<SOME_NAME>"`. Since a JMS Topic or Queue with this name is not deployable on Sun Java System Application Server 7 2004Q2, change this to `"<SOME_NAME>"`, and insert this information in the `sun-ejb-jar.xml`. This change must be reflected for all valid input files, namely, all `.java`, `.jsp` and `.xml` files. Hence, this change of JNDI name is affected globally across the application, and in case of non availability of some source files that contain reference to this jndi-name, you need to make the change manually in them so that the application becomes deployable.

Migrating Sun ONE Application Server 7 Applications

Sun Java System Application Server 7 2004Q2 is an extension to Sun ONE Application Server 7, Enterprise Edition. The following new features are introduced in Application Server 7 2004Q2, Enterprise Edition.

- Stores Stateful Session Beans (SFSB) state and recover it upon planned or unplanned software or hardware failover.
- Integration of Message Queue 3.5
- Automatic retry of failed HTTP requests onto a new server instance.
- Ability to customize the error page displayed when an error occurs due to unavailability of application along HTTP path.
- Supports the following Database Systems:
 - Oracle 9.1
 - DB2 8.1
 - Sybase 12 or late

- MS SQL Server 2000
- Pointbase 4.5 or later

This section discusses the migration issues for the following types of J2EE Applications:

- Migrating Rich Clients
- Migrating EJB Applications

Migrating Rich Clients

This section describes the steps to migrate rich clients from Sun ONE Application Server 7.0 A to Sun Java System Application Server 7 2004Q2. The following scenarios are discussed:

- [Migrating Rich Clients From 7 PE/SE to 7 2004Q2 EE](#)
- [Migrating Rich Clients From 7EE to 7 2004Q2EE](#)

Migrating Rich Clients From 7 PE/SE to 7 2004Q2 EE

Migrating rich clients that are deployed in Sun ONE Application Server 7 PE/SE to Sun Java System Application Server 7 2004Q2 is rather simple. The deployment descriptors used in Sun ONE Application Server 7 can be used as is in Sun Java System Application Server 7 2004Q2. However, if you wish to enable load-balancing and failover features in your client applications, you will need to configure the load-balancing and failover capabilities in the deployment descriptors.

The following procedure describes the steps to migrate the applications:

1. Identify the components which were installed previously.
2. Find out the server-instances, using `asadmin` command or through the directory listing. `asadmin` command requires administration instances to be running. However, administration instances need not be running if the directory listing is used to identify the instances.
3. In the `server.xml` file, add the following `jvm-options` under `jvm-config` element to enable RMI/IIOP failover feature:

```
<jvm-config java-home=path...server-classpath=path>
```

```
<jvm-option>Dorg.omg.PortableInterceptor.ORBInitializerClass.com.sun.ap
pserv.ee.iiop.EEORBInitializer</jvm-option>
```

```
<jvm-option>Dorg.omg.PortableInterceptor.ORBInitializerClass=com.sun.ap
pserv.ee.iiop.EEIORInterceptorInitializer </jvm-option>

<jvm-option>-Dcom.sun.CORBA.connection.ORBSocketFactoryClass=com.sun.ap
pserv.ee.iiop.EEIIOPSocketFactory </jvm-option>

</jvm-config>
```

4. Update the `availability-service` element with `availability-enabled` flag set to `True`:

```
<availability-service availability-enabled="true">

<persistence-store>

<property-name="store-pool-jndi-name" value="" />
<property-name="cluster-id" value="cluster1" />

</persistence-store>

</availability-service>
```

5. Modify the server classpath entry under the `java-config` element to include:

install_dir/SUNWhads/4.2.2-17/lib/hadbjdbc.jar;

install_dir/lib/appserv-rt-ee.jar

6. Add the following `jvm-option` under `java-config` element:

```
<jvm-option>

Dcom.sun.aas.hadbRoot=install-dir/SUNWhadb/4.2.2-17

</jvm-option>
```

7. Update the `sun-acc.xml` with the following new load-balancing properties:

```
<property-name="com.sun.appserv.iiop.loadbalancingpolicy"
value="ic-based" />

<property name="com.sun.appserv.iiop.endpoints"
value=<host>:<port>" />
```

Migrating Rich Clients From 7EE to 7 2004Q2EE

To migrate 7 EE applications to 7 2004Q2 EE, follow the steps given below:

1. Add the following `jvm-options` under `java-config` element for enabling RMI/IIOP failover feature:

```
<jvm-config java-home=path...server-classpath=path>
```



```

<jvm-option>Dorg.omg.PortableInterceptor.ORBInitializerClass.com.sun.ap
pserv.ee.iiop.EEORBInitializer</jvm-option>

<jvm-option>Dorg.omg.PortableInterceptor.ORBInitializerClass.com.sun.ap
pserv.ee.iiop.EEIORInterceptorInitializer </jvm-option>

<jvm-option>-Dcom.sun.CORBA.connection.ORBConnectionFactoryClass=com.sun.ap
pserv.ee.iiop.EEIIOPSocketFactory </jvm-option>

</jvm-config>

```

2. Add the following entry in `server.xml` to setup the iiop-cluster.

```

<iiop-cluster>

    <iiop-server-instance name=<server-name>>

        <iiop-endpoint id=<orb-listener-id>, host=<hostname>,
port=<orb-listener-port>/>

    </iiop-server-instance>

</iiop-cluster>

```

3. Update `sun-acc.xml` with the following new entries:

```

<property-name="com.sun.appserv.iiop.loadbalancingpolicy"
value="ic-based" />

<property name="com.sun.appserv.iiop.endpoints"
value=<host>:<port>" />

```

The host and port is that of ORB-listener-1 's which is defined under `iiop-listener` element in `server.xml`.

Migrating 7 2004Q2, SE Applications to Sun Java System Application Server 7 2004Q2, EE

To migrate the applications from 7 2004Q2 SE, follow the steps described in the section [“Migrating Rich Clients From 7 PE/SE to 7 2004Q2 EE”](#) on page 69.

Migrating EJB Applications to Support SFSB Failover

Sun ONE Application Server 7 does not support failover of stateful session beans. Sun Java System Application Server 7 2004Q2, Enterprise Edition supports failover of stateful session beans on the HTTP and RMI/IIOP path. This section describes the procedure to migrate EJB applications from Sun ONE Application Server 7 SE/PE/EE to Sun Java System Application Server 7 2004Q2 EE to support SFSB state failover.

Migrating EJB Applications From 7.0SE/PE/EE to 7 2004Q2EE

To achieve high availability of EJB applications that use stateful session beans to persist the data, you need to configure a persistent store for each cluster of application servers, where client session information can be maintained across potential failures of individual appserver instances. In addition, the availability-enabled flag must be turned on for each server instance in the cluster.

Sun Java System Application Server supports the failover of stateful session beans. In order to enable this feature in your EJB applications that were deployed to Sun Java System Application Server 7 2004Q2 EE, follow the steps below:

To migrate Entity beans from previous version of Sun's Application Server, follow the procedure described in [“Entity Beans” on page 49](#).

SFSB failover is supported in the following deployment scenario:

When SFSB is accessed from EJBs/Servlets/JSP in J2EE applications executing in the same application server process. The SFSB can be accessed through either a local or remote interface.

In order to take advantage of SFSB state failover support, you need not edit the code. However, you need to provide all the configuration parameters needed for checkpointing the SFSBs in the Sun-specific deployment descriptor (`sun-ejb-jar.xml`) or in the server configuration file.

For detailed information on SFSB Failover, see the *Sun Java System Application Server Administration Guide*.

Installation, Administration, and Deployment

This chapter describes the differences between installing and administering Sun Java™ System Application Server 7 2004Q2 and Sun™ ONE Application Server 6.x Enterprise Edition. A brief description of deployment topologies is also included.

This chapter contains the following sections:

- [Installation differences](#)
- [Administration and Deployment Differences](#)
- [Deployment Topologies](#)

Installation differences

Sun ONE Application Server 6.x, Enterprise Edition had a few supported products that needed to be installed to get the application server up and running. Compared to it, Sun Java System Application Server 7 2004Q2 installation is straight forward as many of the additional components are either built-in or, not required.

The following table provides a description of the differences. The left column lists the various products, the columns in the center and right list the corresponding products as used in Sun ONE Application Server 6.x and Sun Java System Application Server 7 2004Q2, Enterprise Edition, respectively.

Table 4-1 Installation Differences

Products	Sun ONE Application Server 6.x	Sun Java System Application Server 7 2004Q2
Web Server	iPlanet Web Server.	Built-in. An additional web server installation is required for the load balancer plug-in.
LDAP Server	iPlanet Directory Server	Not required.
JMS	iPlanet Message Queue for Java	Built-in (Sun Java System Message Queue).
High-Availability Database (HADB)	Not Required	Required for storing HTTP/S session state.

Minimum Requirements

The following table provides a snapshot of the differences between the requirements of Sun ONE Application Server 6.x and Sun Java System Application Server 7 2004Q2. The left column lists the components, while the center and right columns list the requirements for Sun ONE Application Server 6.x and Sun Java System Application Server 7 2004Q2, Enterprise Edition, respectively.

Table 4-2 Minimum Requirements for the two versions of application servers

Component	Sun ONE ApplicationN Server 6.x	Sun Java System Application Server 7 2004Q2
Operating System	<ul style="list-style-type: none">• Solaris 2.6, 8, 9 (SPARC)• HP-UX• IBM AIX• Windows NT, Windows 2000	Solaris 8, 9 (SPARC) Solaris 9 (X86) Red Hat Linux 7.2 Red Hat Enterprise Linux 2.1 Windows 2000, 200 SP2, XP
Hard Disk space	400 MB	250 MB
RAM	512 MB	512 MB
J2SE	1.3.1_06 (iPlanet Application Server 6.5, SP1, Enterprise Edition)	1.4.0_02

Table 4-2 Minimum Requirements for the two versions of application servers

Component	Sun ONE Application Server 6.x	Sun Java System Application Server 7 2004Q2
Web server	<ul style="list-style-type: none"> iPlanet/Sun ONE Web Server 6.0, SP1, SP2, or 6.0.1, iPlanet Web Server 4.7 Apache 1.3.19, 1.13.26 Microsoft IIS 4.0, 5.0 	<ul style="list-style-type: none"> Sun ONE Web Server 6.0 SP2 Apache 1.3.27 Microsoft IIS 4.0, 5.0
Directory Server	iPlanet/Sun ONE Directory Server 5.0	Not required
High-Availability Database	Not available	Supplied with product.

Installation Procedure differences

- Unlike Sun ONE Application Server 6.x, you do not have to install iPlanet Web Server or iPlanet Directory Server before installing Sun Java System Application Server 7 2004Q2. However, you will need a separate web server to install the load balancer plug-in.
- Before you install Sun Java System Application Server 7 2004Q2, Enterprise Edition software, you will need to decide on product topology of the application server and the High-Availability Database (HADB) server.

They can generally be hosted in two ways:

- Application server and HADB server node hosted on the same system
- Application server and HADB server node hosted on separate systems

In both cases, at least two systems per component are needed to achieve high availability.

NOTE Only 8-16 connections are allowed per HADB node. To take care of high loads, you should plan to add more HADB nodes.

Details on the various topologies that can be implemented for the Always On Technology are discussed in the Enterprise Edition of *Sun Java System Application Server, System Deployment Guide*.

For detailed instructions for installing the Application Server software, see the *Sun Java System Application Server Installation Guide*.

- Sun ONE Application Server 6.x supported various load balancing schemes, along with the web server plug-in and application server driven mechanism. Additionally, the web server plug-in driven method supported a wide variety of sticky round robin scheme.

For more information on these load balancing methods, see *Sun ONE Application Server 6.x, Enterprise Edition, Administrator's Guide*.

Sun Java System Application Server 7 2004Q2, Enterprise Edition supports a simple, round robin scheme, with sophisticated health checking and failover capabilities. For more information on the load balancing schemes, see the *Sun Java System Application Server Administration Guide*.

Administration and Deployment Differences

The following table lists the different administration and deployment utilities between the two application server versions.

Table 4-3 Administration and Deployment Differences

Products	Sun ONE Application Server 6.x	Sun Java System Application Server 7 2004Q2
Administration Interface	Administration Server.	Administration Server.
	Location: <i>install_dir</i> /ias/bin	Location: <i>install_dir</i> /domains/ <i>domain_dir</i> /admin-server/bin
	GUI: Console based Administration Tool (iASAT).	GUI: Browser based administration tool.
	Command line too: iascontrol	Command line too: asadmin
Deployment tool	Separate deployment tool	Sun ONE Studio for Java 5, Update 1 or asadmin.
	GUI: Console based deployment tool	Assembly Tool
	Command line tool: iasdeploy	
Registry Editor	GUI: iPlanet Registry Editor (kregedit).	Not required.
cladmin	Not available	Command line tool for cluster administration.
clsetup	Not Available	Sets up HTTP/S session persistence and Application Server in a typical cluster environment with the most common working requirements.

Table 4-3 Administration and Deployment Differences

Products	Sun ONE Application Server 6.x	Sun Java System Application Server 7 2004Q2
Load balancing	Administration Tool	Configuration file based. The loadbalancer.xml file contains the configurable parameters.
Session failover	Administration Tool	HADB command line utility

Non-root Installation and Administration

Sun Java System Application Server 7 2004Q2, Enterprise Edition does not require root privileges to install. For details see, *Sun Java System Application Server Installation Guide*.

Deployment Topologies

For various, detailed Sun Java System Application Server 7 2004Q2 deployment topologies, see the *Sun Java System Application Server System Deployment Guide*.

Migrating iBank Application - Walkthrough

This chapter describes the process for migrating the main components of a typical J2EE application from Sun ONE Application Server 6.0 and 6.5 to Sun Java System Application Server 7 2004Q2. This chapter highlights the problems posed during the migration of each type of component and suggests practical solutions to overcome such problems.

For this migration process, the J2EE application presented is called iBank and is based on the actual migration of the iBank application from the Sun ONE Application Server 6.0 and 6.5 versions to Sun Java System Application Server 7 2004Q2. iBank simulates an online banking service and covers all of the aspects traditionally associated with a J2EE application.

The sensitive points of the J2EE specification covered by the iBank application are summarized below:

- Servlets, especially with redirection to JSP pages (model-view-controller architecture)
- JSP pages, especially with static and dynamic inclusion of pages
- JSP custom tag libraries
- Creation and management of HTTP sessions
- Database access through the JDBC API
- Enterprise JavaBeans: Stateful and Stateless session beans, CMP and BMP entity beans
- Assembly and deployment in line with the standard packaging methods of the J2EE application

The iBank application is presented in detail in [Appendix A, “iBank Application Specification.”](#)

NOTE The sample application, *iBank*, is not a clustered application - because of which this chapter does not discuss about the migration issues related to clustered applications.

Preparing for Migrating the iBank Application

Before you start with the migration process learn about the differences in the deployment descriptors. For detailed information, see [“Migrating Deployment Descriptors” on page 67.](#)

Choosing the Target

If your applications support failover capabilities, you must choose the migration target server as Sun Java System Application Server 7 2004Q2, Enterprise Edition. The persistence type *ha* is the persistence type supported for production environments that require failover capabilities in Sun Java System Application Server 7 2004Q2, Enterprise Edition. For more details on enabling failover capabilities to your migrated applications, see the *Sun Java System Application Server Administration Guide*.

If your applications do not use the high availability features, you may choose to migrate your applications to Sun Java System Application Server 7 2004Q2, Standard Edition.

After choosing the target server, install the server on your migration environment. For step-by-step instructions to install the software, see the *Sun Java System Application Server Installation Guide*.

If you are using Sun Java System Migration Tool to migrate the components, you must install the tool. The Sun Java System Migration Tool can be downloaded from the following location:

<http://www.sun.com/migration>

For information on how to use the tool, see the Sun Java System Migration Tool online help.

Development Environment

The iBank application in Sun ONE Application Server 6.5 is developed using the Sun ONE Application Server 6.5 and Sun ONE Studio. Sun ONE Application Server 6.5 supports the following features:

- Load-balancing of applications across server instances
- Failover of server instances nodes

Identifying the Components of the iBank Application

- Extract the application, which is in a zip format in a local directory.

The source for the iBank application (iBank65.zip) can be found at the migration site <http://www.sun.com/migration/sunonetools.html>. Unzipping the file iBank65.zip would create the following directory structure:

```
iBank
/docroot
/session
/entity
/misc
```

- /docroot contains HTML, JSP's and Image files in its root. It also contains the source files for servlets and EJBs in the sub-folder WEB-INF\classes following the package structure com.sun.bank.*. A war file is generated through the contents of this directory.
- /session contains the source code for the session beans following the package structure com.sun.bank.ejb.session. This directory forms the EJB module for the session beans.
- /entity contains the entity beans following the package structure com.sun.bank.ejb.entity. This directory would form the EJB module for entity beans.
- /misc contain the sql scripts for the database setup.

Setup the Schema

Setup the schema for iBank application by executing the sql scripts provided in the /misc folder. These scripts are for oracle database. These scripts creates user, creates tables and insert data into the tables. Execute the scripts in the following order:

- 01_iBank_CreateUser.sql
- 02_iBank_CreateTables.sql
- 03_iBank_InsertData.sql

Manual Migration of iBank Application

The manual migration does not require any major changes in the source code as Sun Java System Application Server 7 2004Q2 supports CMP 1.1. However, manual migration of the application would require a few changes to be made in the following aspects:

Web Application Changes

Migrating iBank from Sun ONE Application server 6.0/6.5 to Sun Java System Application Server 7 2004Q2 would not require any changes in the web application part of the iBank application. Delete the `ias-web.xml` file from the source directory, as there is no information in this file that can go inside its counterpart in the Sun Java System Application Server 7 2004Q2 deployment descriptor, the `sun-web.xml` file. The `web.xml` requires no changes.

However, generically speaking, if there is some information inside the `web.xml` that needs to be mapped to the Server specific resources, then a declaration in `sun-web.xml` will be required. For example, if the `web.xml` file had declared a `javax.sql.DataSource` type resource reference, it is mandatory to map it to the JNDI name of the actual `DataSource` on the Server, inside the `sun-web.xml`.

You need to create the new `sun-web.xml`. The process of creating Sun Java System Application Server specific deployment descriptor `sun-web.xml` is outlined below:

1. Create a new XML file which has the following DOCTYPE definition on top:

```
'http://www.sun.com/software/sunone/appserver/dtds/sun-web-app_2_3-1.dtd'
```

Save this file as `sun-web.xml`.

2. The root tag of this XML file, as evident from the DOCTYPE definition, is `sun-web`. In the DTD, this element is defined as follows:

```
<!ELEMENT sun-web-app (security-role-mapping*, servlet*,
session-config?, resource-env-ref*, resource-ref*, ejb-ref*,
cache?, class-loader?, jsp-config?, locale-charset-info?,
property*)>
```

From the above declaration it is clear that all tags are optional. Thus, a default `sun-web.xml` would look something like:

```
<!DOCTYPE sun-web-app SYSTEM
"http://www.sun.com/software/sunone/appserver/dtds/sun-web-app_2_3-1.dtd">

</sun-web-app>
```

3. To declare any resource references, use the following element declaration:

```
<!ELEMENT resource-ref (res-ref-name, jndi-name,  
default-resource-principal?)> where the sub elements are:
```

```
<!ELEMENT res-ref-name (#PCDATA)>  
<!ELEMENT default-resource-principal (name, password)>  
<!ELEMENT jndi-name (#PCDATA)>
```

In case of iBank application `sun-web.xml` must contain the following resource-reference declarations:

```
<sun-web-app>  
  
  <resource-ref>  
  
    <res-ref-name>jdbc/IBank</res-ref-name>  
  
    <jndi-name>jdbc/IBank</jndi-name>  
  
    <default-resource-principal>  
  
      <name>ibank_user</name>  
  
      <password>ibank_user</password>  
  
    </default-resource-principal>  
  
  </resource-ref>  
  
</sun-web-app>
```

EJB Changes

Migrating iBank from Sun ONE Application Server 6.5 to Sun Java System Application Server 7 2004Q2 does not require any changes in the EJB code.

Session Beans:

In `ejb-jar.xml`: The `<!DOCTYPE` definition should be modified to point to the latest DTD url in case of `ejb-jar.xml`. This new definition should look like this:

```
<!DOCTYPE ejb-jar PUBLIC "-//Sun Microsystems, Inc.//DTD Enterprise  
JavaBeans 2.0//EN"  
'http://java.sun.com/dtd/ejb-jar_2_0-1.dtd'>
```

The `ias-ejb-jar.xml` in Sun ONE Application server 6.5 has been replaced by `sun-ejb-jar.xml` in Sun Java System Application server 7 2004Q2. Since the DTDs for these two XML files are radically different, you need to create the new `sun-ejb-jar.xml` by extracting relevant information from the `ejb-jar.xml` and `ias-ejb-jar.xml`. The process of creating the `ejb-jar.xml` file is outlined below:

1. Create a new XML file which has the following DOCTYPE definition on top:

```
<!DOCTYPE sun-ejb-jar PUBLIC "-//Sun Microsystems, Inc.//DTD Sun ONE
Application Server 7.1 EJB 2.0//EN"
'http://www.sun.com/software/sunone/appserver/dtds/sun-ejb-jar_2_0-
1.dtd'>
```

Save this file as “sun-ejb-jar.xml”, along with the modified ejb-jar.xml.

2. The root tag of this XML file, as evident from the DOCTYPE definition, is sun-ejb-jar. In the DTD, this element is defined as

```
<!ELEMENT sun-ejb-jar (security-role-mapping*,enterprise-beans)>
```

The `security-role-mapping` tag is meant for mapping the security roles declared in the `ejb-jar.xml`. Since there is no security declared in the `ejb-jar.xml` file of the iBank application, you can skip the declaration of this tag. Focus on the `enterprise-beans` tag. Right now, the `sun-ejb-jar.xml` file should have the following contents:

```
<sun-ejb-jar>
    <enterprise-beans>
    </enterprise-beans>
</sun-ejb-jar>
```

NOTE: The header part of the document, namely the XML declaration and DOCTYPE definition, are not included here for brevity.

3. The `enterprise-beans` element is defined in the DTD as follows:

```
<!ELEMENT enterprise-beans (name?, unique-id?, ejb*,
pm-descriptors?, cmp-resource?)>
```

The optional `name` element should contain the canonical name of the `<enterprise-beans>`. You may give it some name.

The `<unique-id>` element is used by the Sun Java System Application Server and is inserted by the Application Server automatically at the time of application deployment.

The `EJB` element tag is the most important tag. This element describes the runtime bindings for a single EJB. It is defined in the DTD as follows:

```
<!ELEMENT ejb (ejb-name, jndi-name?, ejb-ref*, resource-ref*,
resource-env-ref*, pass-by-reference?, cmp?, principal?,
mdb-connection-factory?, jms-durable-subscription-name?,
jms-max-messages-load?, ior-security-config?, is-read-only-bean?,
refresh-period-in-seconds?, commit-option?, gen-classes?,
bean-pool?, bean-cache?)>
```

In the sample, the `ejb` element will contain the `ejb-name` element. The `ejb-name` element will contain the canonical name of the EJB. This name should be the same as declared inside the `ejb-name` element of the `ejb-jar.xml` for that EJB. It will also contain the `jndi-name` of the EJB. One of the differences between Sun ONE Application Server 6.5 and Sun Java System Application Server 7 2004Q2 is the flexibility of the latter in providing freedom to the bean developer to have different `ejb-name` and `jndi-name` of an EJB. In Sun ONE Application Server 6.5, the `jndi-name` of an EJB by default was `ejb/<ejb-name>`.

To allow smooth migration, keep the `jndi-names` of the EJB and all other resources to be same as they were on Sun ONE Application Server 6.5. Hence, declare the *ejb-name* of all the EJBs' to be `ejb/<ejb-name>`.

Using the logic described above, the `sun-ejb-jar.xml` now should have the following contents:

```
<sun-ejb-jar>

<enterprise-beans>

<ejb>

<ejb-name>BankTeller</ejb-name>

<jndi-name>ejb/BankTeller</jndi-name>

</ejb>

<ejb>

<ejb-name>InterestCalculator</ejb-name>

<jndi-name>ejb/InterestCalculator</jndi-name>

</ejb>

</enterprise-beans>

</sun-ejb-jar>
```

4. For each `<ejb-ref>` element inside the `ejb-jar.xml`, there should be a corresponding `<ejb-ref>` element inside the `sun-ejb-jar.xml`. The `<ejb-ref>` element inside the `ejb-jar.xml` is used to declare all the EJBs referenced from inside the bean class of that EJB. While the bean class code will reference the EJB by using its `<ejb-ref-name>`, this `<ejb-ref-name>` has to be mapped to the actual `<jndi-name>` of the bean on the Application Server. Hence, this serves as a mechanism to add a layer of abstraction between the name referenced by the EJB implementation and the actual JNDI name of the bean.

Using the logic explained above, consider examining the `BankTeller` EJB. In the `ejb-jar.xml`, there are two `<ejb-ref>` declarations inside this EJB. The first one is for the `Customer` EJB (an entity bean in the Entity Bean module). As explained in Step 3 above, the JNDI names of all EJBs must be kept as `ejb/<ejb-name>`, and add this declaration inside the `sun-ejb-jar.xml`

```
<sun-ejb-jar>

<enterprise-beans>

<ejb>

  <ejb-name>BankTeller</ejb-name>
  <jnndi-name>ejb/BankTeller</jndi-name>

  <ejb-ref>

    <ejb-ref-name>Customer</ejb-ref-name>
    <jndi-name>ejb/Customer</jndi-name>

  </ejb-ref>

</ejb>

<ejb>

  <ejb-name>InterestCalculator</ejb-name>
  <jndi-name>ejb/InterestCalculator</jndi-name>

</ejb>

</enterprise-beans>

</sun-ejb-jar>
```


Similarly, add a similar `<ejb-ref>` tag for Account EJB. Since the InterestCalculator bean does not have a `<ejb-ref>` tag inside the `ejb-jar.xml`, it is not required inside the `sun-ejb-jar.xml` also. Now, the `sun-ejb-jar.xml` has the following contents:

```
<sun-ejb-jar>

<enterprise-beans>

<ejb>

  <ejb-name>BankTeller</ejb-name>
  <jndi-name>ejb/BankTeller</jndi-name>

  <ejb-ref>

    <ejb-ref-name>Customer</ejb-ref-name>
    <jndi-name>ejb/Customer</jndi-name>

  </ejb-ref>

  <ejb-ref>

    <ejb-ref-name>Account</ejb-ref-name>
    <jndi-name>ejb/Account</jndi-name>

  </ejb-ref>

</ejb>

<ejb>

  <ejb-name>InterestCalculator</ejb-name>
  <jndi-name>ejb/InterestCalculator</jndi-name>

</ejb>

</enterprise-beans>

</sun-ejb-jar>
```

5. The `ejb` element would contain element pass-by-reference `<!ELEMENT pass-by-reference (#PCData)`.

`pass-by-reference` elements control the use of Pass by Reference semantics. The EJB specification requires pass by value, which will be the default mode of operation. This can be set to true for non-compliant operation and possibly higher performance. It can apply to all the enclosed EJB modules. Allowed values are true and false. Default value is false.

6. The `ejb` element also has an element named `<bean-cache>`.

```
<!ELEMENT bean-cache (max-cache-size?, is-cache-overflow-allowed?,
cache-idle-timeout-in-seconds?, removal-timeout-in-seconds?,
victim-selection-policy?)>
```

This element is used only for the stateful session beans and the entity beans. In the iBank application, only `BankTeller` session bean will have this entry.

In this tag, *max-cache-size* defines the maximum number of beans in the cache. *cache-idle-timeout-in-seconds* specifies the maximum time that a stateful session bean or entity bean is allowed to be idle in the cache. After this time, the bean is passivated to backup store. This is a hint to the server. Default value for *cache-idle-timeout-in-seconds* is 10 minutes.

The amount of time that the bean remains passivated (i.e. idle in the backup store) is controlled by *removal-timeout-in-seconds* parameter. Note that if a bean was not accessed beyond *removal-timeout-in-seconds*, then it will be removed from the backup store and hence will not be accessible to the client. The Default value for *removal-timeout-in-seconds* is 60min.

With the above entries, `sun-ejb-jar.xml` should look like this:

```
<sun-ejb-jar>
<enterprise-beans>
<ejb>
<ejb-name>BankTeller</ejb-name>
<jndi-name>ejb/BankTeller</jndi-name>
<ejb-ref>
<ejb-ref-name>Customer</ejb-ref-name>
<jndi-name>ejb/Customer</jndi-name>
</ejb-ref>
<ejb-ref>
<ejb-ref-name>Account</ejb-ref-name>
<jndi-name>ejb/Account</jndi-name>
</ejb-ref>
<pass-by-reference>false</pass-by-reference>
<bean-cache>
```

```

<cache-idle-timeout-in-seconds>
0
</cache-idle-timeout-in-seconds>
<removal-timeout-in-seconds>
0
</removal-timeout-in-seconds>
</bean-cache>
</ejb>
<ejb>
<ejb-name>InterestCalculator</ejb-name>
<jndi-name>ejb/InterestCalculator</jndi-name>
<pass-by-reference>false</pass-by-reference>
</ejb>
</enterprise-beans>
</sun-ejb-jar>

```

7. The element used only for defining the stateless session bean and the message-driven bean pool is the `<bean-pool>`.

```

<!ELEMENT bean-pool (steady-pool-size?, resize-quantity?,
max-pool-size?, pool-idle-timeout-in-seconds?,
max-wait-time-in-millis?)>

```

steady-pool-size specifies the initial and minimum number of beans that must be maintained in the pool.

resize-quantity specifies the number of beans to be created or deleted when the pool is being serviced by the pool manager.

max-pool-size specifies the maximum pool size. Valid values are from 0 to MAX_INTEGER.

max-pool-size spiffiest the maximum pool size.

pool-idle-timeout-in-seconds specifies the maximum time that a stateless session bean or message-driven bean is allowed to be idle in the pool.

Finally the `sun-ejb-jar.xml` will have the following contents:

```

<sun-ejb-jar>
  <enterprise-beans>
    <ejb>
      <ejb-name>BankTeller</ejb-name>
      <jndi-name>ejb/BankTeller</jndi-name>
      <ejb-ref>
        <ejb-ref-name>Customer</ejb-ref-name>
        <jndi-name>ejb/Customer</jndi-name>
      </ejb-ref>
      <ejb-ref>
        <ejb-ref-name>Account</ejb-ref-name>
        <jndi-name>ejb/Account</jndi-name>
      </ejb-ref>
      <pass-by-reference>false</pass-by-reference>
      <bean-cache>
        <cache-idle-timeout-in-seconds>
          0
        </cache-idle-timeout-in-seconds>
        <removal-timeout-in-seconds>
          0
        </removal-timeout-in-seconds>
      </bean-cache>
    </ejb>
    <ejb>
      <ejb-name>InterestCalculator</ejb-name>
      <jndi-name>ejb/InterestCalculator</jndi-name>
      <pass-by-reference>false</pass-by-reference>
      <bean-pool>
        <pool-idle-timeout-in-seconds>

```

```

0
    </pool-idle-timeout-in-seconds>
  </bean-pool>
</ejb>
</enterprise-beans>
</sun-ejb-jar>

```

Entity Beans:

In `ejb-jar.xml`: The `<!DOCTYPE` definition should be modified to point to the latest DTD url in case of `ejb-jar.xml`. This new definition should look like this:

```

<!DOCTYPE sun-ejb-jar PUBLIC "-//Sun Microsystems, Inc.//DTD Sun ONE
Application Server 7.1 EJB 2.0//EN"
'http://www.sun.com/software/sunone/appserver/dtds/sun-ejb-jar_2_0-1.dtd'>

```

Insert `<cmp-version>` tag with value 1.1 for all CMPs beans in `ejb-jar.xml`.

Entry for entity bean would look like:

```

<entity>
  <description>Account CMP entity bean</description>
  <ejb-name>Account</ejb-name>
  <home>com.sun.bank.ejb.entity.AccountHome</home>
  <remote>com.sun.bank.ejb.entity.Account</remote>
  <ejb-class>com.sun.bank.ejb.entity.AccountEJB</ejb-class>
  <persistence-type>Container</persistence-type>
  <prim-key-class>
    com.sun.bank.ejb.entity.AccountPK
  </prim-key-class>
  <reentrant>False</reentrant>
  <cmp-version>1.x</cmp-version>
<cmp-field>
  <field-name>branchCode</field-name></cmp-field>
<cmp-field>
  <field-name>accTypeId</field-name></cmp-field>

```

```

    <cmp-field>
        <field-name>accBalance</field-name></cmp-field>
    <cmp-field>
        <field-name>custNo</field-name></cmp-field>
    <cmp-field>
        <field-name>accNo</field-name></cmp-field>
</entity>

```

Similarly, all the CMP beans would have this entry.

Similar to Session Beans, the `ias-ejb-jar.xml` in Sun ONE Application Server 6.5 has been replaced by `sun-ejb-jar.xml` in Sun Java System Application Server 7 2004Q2. Since the DTDs for these two XML files are radically different, you need to create the new `sun-ejb-jar.xml` by extracting relevant information from the `ejb-jar.xml` and `ias-ejb-jar.xml`. The process of creating the `sun-ejb-jar.xml` is outlined below:

1. Create a new XML file which has the following DOCTYPE definition on top:

```

<!DOCTYPE sun-ejb-jar PUBLIC "-//Sun Microsystems, Inc.//DTD Sun ONE
Application Server 7.1 EJB 2.0//EN"
'http://www.sun.com/software/sunone/appserver/dtds/sun-ejb-jar_2_0-
1.dtd'>

```

Save this file as “`sun-ejb-jar.xml`”, along with the modified `ejb-jar.xml`.

2. The root tag of this XML file, as evident from the DOCTYPE definition, is `sun-ejb-jar`. In the DTD, this element is defined as

```

<!ELEMENT sun-ejb-jar (security-role-mapping*, enterprise-beans) >

```

The `security-role-mapping` tag is meant for mapping the security roles declared in the `ejb-jar.xml`. As in the iBank application, there is no security declared in the `ejb-jar.xml` file, skip the declaration of `security-role-mapping` optional tag and focus on the `enterprise-beans` tag. The `sun-ejb-jar.xml` file should have the following contents.

```

<sun-ejb-jar>
    <enterprise-beans> </enterprise-beans>
</sun-ejb-jar>

```

NOTE: The header part of the document, namely the XML declaration and DOCTYPE definition are not included here for brevity.

3. The `<enterprise-beans>` element is defined in the DTD as follows:

```
<!ELEMENT enterprise-beans (name?, unique-id?, ejb*,  
pm-descriptors?, cmp-resource?)>
```

The optional `name` element should contain the canonical name of the `<enterprise-beans>`. You may give it some name.

The `<unique-id>` element is used by Sun Java System Application Server and is inserted by the Application Server automatically at the time of application deployment.

The `ejb` element describes the runtime bindings for a single EJB. It is defined in the DTD as follows:

```
<!ELEMENT ejb (ejb-name, jndi-name?, ejb-ref*, resource-ref*,  
resource-env-ref*, pass-by-reference?, cmp?, principal?,  
mdb-connection-factory?, jms-durable-subscription-name?,  
jms-max-messages-load?, ior-security-config?, is-read-only-bean?,  
refresh-period-in-seconds?, commit-option?, gen-classes?,  
bean-pool?, bean-cache?)>
```

In this case, the `ejb` element will contain the `ejb-name` element. The `ejb-name` element will contain the canonical name of the EJB. This name should be the same as declared inside the `ejb-name` element of the `ejb-jar.xml` for that EJB. It will also contain the `jndi-name` of the EJB. One of the differences between Sun ONE Application Server 6.5 and Sun Java System Application Server 7 2004Q2 is the flexibility of the latter in providing freedom to the bean developer to have different `ejb-name` and `jndi-name` of an EJB. In Sun ONE Application Server 6.5, the JNDI name of an EJB by default was `ejb/<ejb-name>`.

To allow smooth migration, keep the `jndi-names` of the EJB and all other resources to be same as they were on Sun ONE Application Server 6.5. Declare the `ejb-name` of all the `ejbs` to be `ejb/<ejb-name>`.

Using the logic described above, the `sun-ejb-jar.xml` now should have the following entries:

```
<sun-ejb-jar>
<enterprise-beans>

<ejb>
<ejb-name> Account</ejb-name>
<jndi-name> ejb/Account</jndi-name>
</ejb>
<ejb> --- </ejb>
<ejb> --- </ejb>
other ejb's
<ejb> --- </ejb>
<ejb> --- </ejb>
</enterprise-beans>
</sun-ejb-jar>
```

4. The `ejb` element contains the element `pass-by-reference` `<!ELEMENT pass-by-reference (#PCData)`.

`pass-by-reference` elements control the use of Pass by Reference semantics. EJB specification requires pass by value, which will be the default mode of operation. This can be set to true for non-compliant operation and possibly higher performance. It can apply to all the enclosed EJB modules. Allowed values are true and false. Default value is false.

5. In case of CMP entity beans, element `cmp` is declared, which describes runtime information for a CMP EntityBean object for EJB1.1 and EJB2.0 beans.

```
<!ELEMENT cmp (mapping-properties?, is-one-one-cmp?,  
one-one-finders?)>
```

The *mapping-properties* tag contains the location of the persistence vendor specific O/R mapping file. *is-one-one-cmp* field is used to identify CMP 1.1 with old descriptors. This contains the boolean true if it is CMP 1.1. *one-one-finders* contains the finders for CMP 1.1.

This root element `<finder>` contains the finder for CMP 1.1 with a `method-name` and query parameters.

```
<!ELEMENT finder (method-name, query-params?, query-filter?,  
query-variables?)>
```

Element *method-name* contains the method name for the query field. Element *query-params* contains the query parameters for CMP 1.1 finder.

query-filter is an optional element which contains the query filter for CMP 1.1 finder.

After making the above entries in `iBank`, `sun-ejb-jar.xml` would look like the following:

```
<sun-ejb-jar>  
<enterprise-beans>  
<ejb>  
<ejb-name> Account</ejb-name>  
<jndi-name> ejb/Account</jndi-name>  
<pass-by-reference>false</pass-by-reference>  
<cmp>  
<mapping-properties>  
META-INF/sun-cmp-mappings.xml  
</mapping-properties>  
<is-one-one-cmp>true</is-one-one-cmp>  
<one-one-finders>
```

```

<finder>
<method-name>
findOrderedAccountsForCustomer
</method-name>
<query-params>int custNo</query-params>
<query-filter>
custNo == custNo
</query-filter>
</finder>
</one-one-finders>
</cmp>
</ejb>
<ejb> --- </ejb>
<ejb> --- </ejb>
other ejb's
<ejb> --- </ejb>
<ejb> --- </ejb>
</enterprise-beans>
</sun-ejb-jar>

```

Account is the only entity bean having a finder other than primary key. Hence the finder entry shown above would only be in the case of Account bean.

6. The `<!ELEMENT commit-option (#PCDATA)>` specifies option for committing.

7. The `<ejb>` element also has an element `<bean-cache>`.

```
<!ELEMENT bean-cache (max-cache-size?, is-cache-overflow-allowed?,
cache-idle-timeout-in-seconds?, removal-timeout-in-seconds?,
victim-selection-policy?)>
```

This element is used only for stateful session beans and entity beans. In this tag, *max-cache-size* defines the maximum number of beans in the cache. *cache-idle-timeout-in-seconds* specifies the maximum time that a stateful session bean or an entity bean is allowed to be idle in the cache. After this time, the bean is passivated to backup store. This is a hint to the server. Default value for *cache-idle-timeout-in-seconds* is 10 minutes.

The amount of time that the bean remains passivated (i.e. idle in the backup store) is controlled by *removal-timeout-in-seconds* parameter. Note that if a bean was not accessed beyond *removal-timeout-in-seconds*, then it will be removed from the backup store and hence will not be accessible to the client. The Default value for *removal-timeout-in-seconds* is 60 min.

With the above entries, `sun-ejb-jar.xml` should look like this:

```
<sun-ejb-jar>

<enterprise-beans>

<ejb>

<ejb-name> Account</ejb-name>
<jndi-name> ejb/Account</jndi-name>
<pass-by-reference>false</pass-by-reference>

<cmp>

<mapping-properties>

META-INF/sun-cmp-mappings.xml

</mapping-properties>

<is-one-one-cmp>true</is-one-one-cmp>

<one-one-finders>

<finder>

<method-name>

findOrderedAccountsForCustomer

</method-name>
```

```

<query-params>int custNo</query-params>
<query-filter>
custNo == custNo
</query-filter>
</finder>
</one-one-finders>
</cmp>
<commit-option>C</commit-option>
<bean-cache>
<max-cache-size>60</max-cache-size>
<cache-idle-timeout-in-seconds>0</cache-idle-timeout-in-seconds>
</bean-cache>
</ejb>
<ejb> --- </ejb>
<ejb> --- </ejb>
other ejb's
<ejb> --- </ejb>
<ejb> --- </ejb>
</enterprise-beans>
</sun-ejb-jar>

```

8. In `<!ELEMENT enterprise-beans (name?, unique-id?, ejb*, pm-descriptors?, cmp-resource?)>`

Element `<pm-descriptors>` would be as follows:

```
<!ELEMENT pm-descriptors (pm-descriptor+, pm-inuse)>
```

Persistence Manager descriptors contain one or more `pm-descriptors`, but only one of them must be in use at any given time.

`<pm-descriptor>` describes the properties for the persistence manager associated with entity bean. *pm-identifier* element describes the vendor who provided the PM implementation. *pm-version* further specifies which version of PM vendor product to be used. *pm-config* specifies the vendor specific config file to be used. *pm-class-generator* specifies the vendor specific concrete class

generator. This is the name of the class specific to a vendor. *pm-mapping-factory* specifies the vendor specific mapping factory. This is the name of the class specific to a vendor. *pm-insue* specifies whether this particular PM must be used or not.

The element `<cmp-resource>` contains the database to be used for storing CMP beans in an ejb-jar.

```
<!ELEMENT cmp-resource (jndi-name, default-resource-principal?)>
```

The element *jndi-name* specifies the JNDI name string. Element *default-resource-principal* has element name and password to be used when none are specified while accessing a resource.

```
<!ELEMENT default-resource-principal ( name, password)>
```

Finally `sun-ejb-jar.xml` should contain the following:

```
<sun-ejb-jar>
  <enterprise-beans>
    <ejb>
      <ejb-name> Account</ejb-name>
      <jndi-name> ejb/Account</jndi-name>
      <pass-by-reference>false</pass-by-reference>
      <cmp>
        <mapping-properties>
          META-INF/sun-cmp-mappings.xml
        </mapping-properties>
        <is-one-one-cmp>true</is-one-one-cmp>
        <one-one-finders>
          <finder>
            <method-name>
              findOrderedAccountsForCustomer
            </method-name>
            <query-params>int custNo</query-params>
            <query-filter>
              custNo == custNo
            </query-filter>
          </finder>
        </one-one-finders>
      </cmp>
    </ejb>
  </enterprise-beans>
</sun-ejb-jar>
```

```

        </query-filter>
    </finder>
</one-one-finders>
</cmp>
<commit-option>C</commit-option>
<bean-cache>
    <max-cache-size>60</max-cache-size>
    <cache-idle-timeout-in-seconds>
        0
    </cache-idle-timeout-in-seconds>
</bean-cache>
</ejb>

<ejb> --- </ejb>
<ejb> --- </ejb>

                                other ejb's

<ejb> --- </ejb>
<ejb> --- </ejb>
<pm-descriptors>
    <pm-descriptor>
        <pm-identifier>IPLANET</pm-identifier>
        <pm-version>1.0</pm-version>
        <pm-class-generator>
            com.iplanet.ias.persistence.
            internal.ejb.ejbci.JDOCodeGenerator
        </pm-class-generator>
        <pm-mapping-factory>
            com.iplanet.ias.cmp.NullFactory
        </pm-mapping-factory>
    </pm-descriptor>

```

```

        <pm-inuse>
        <pm-identifier>IPLANET</pm-identifier>
        <pm-version>1.0</pm-version></pm-inuse>
    </pm-descriptors>
    <cmp-resource>
    <jndi-name>jdo/pmf</jndi-name>
    </cmp-resource>
</enterprise-beans>
</sun-ejb-jar>

```

Generate dbschema by using the *capture-schema* utility in Sun Java System Application Server 7 2004Q2 installation's bin directory. Execute `capture-schema.bat` file stored in the `\bin` directory and specify the valid inputs for the database URL, username, password, and specify the tables for which schema has to be generated. By default, schema has to be generated for all the tables used by the application. In case of iBank, there are six tables for which schema has to be generated. Name this schema file as `myschema.dbschema`. The tables used in iBank are:

```

ACCOUNT
ACCOUNT_TYPE
BRANCH
CUSTOMER
TRANSACTION_HISTORY
TRANSACTION_TYPE

```

Store the file `myschema.dbschema` above the `META-INF` folder for the entity beans.

In `<ejb-name>-ias-cmp.xml`: Replace all the `<ejb-name>-ias-cmp.xml` files in Sun ONE Application Server 6.0/6.5 with one `sun-cmp-mappings.xml` file. This file maps (at least one) set of beans to tables and columns in a specific dbschema. Since the DTDs for this two XML files are radically different, You need to actually create a new file following the steps given below:

1. Create a new XML file which has the following DOCTYPE definition on top:

```
<!DOCTYPE sun-cmp-mappings PUBLIC "-//Sun Microsystems, Inc.//DTD  
Sun ONE Application Server 7.0 OR Mapping //EN"  
'http://www.sun.com/software/sunone/appserver/dtds/sun-cmp_mapping_  
1_0.dtd'>
```

Save this file as “sun-cmp-mappings.xml”.

2. The root tag of this XML file, as evident from the DOCTYPE definition, is sun-cmp-mappings. In the DTD, this element is defined as follows:

```
<!ELEMENT sun-cmp-mappings ( sun-cmp-mapping+ ) >
```

Element sun-cmp-mapping would be :

```
<!ELEMENT sun-cmp-mapping ( schema, entity-mapping+ ) >
```

Here element schema is the path name to the schema file.

A CMP bean has a name, a primary table, one or more fields, zero or more relationships, and zero or more secondary tables, plus flags for consistency checking. Element entity-mapping has the following elements:

```
<!ELEMENT entity-mapping (ejb-name, table-name, cmp-field-mapping+,  
cmr-field-mapping*, secondary-table*, consistency?)>
```

Element *ejb-name* is the EJB name from standard ejb-jar DTD. Element *table-name* is the name of the database table. A *cmp-field-mapping* has a field, one or more columns that it maps to *cmr-field mapping*. A *cmr* field has a name and one or more column pairs that define the relationship. Element *secondary-table* is for secondary table used. In case of iBank, no secondary table is used.

After making the changes described above, the sun-cmp-mappings.xml file with entries for Account entity bean should look like the following:

```
<sun-cmp-mapping>  
  <schema>mySchema</schema>  
  <entity-mapping>  
    <ejb-name>Account</ejb-name>  
    <table-name>ACCOUNT</table-name>  
    <cmp-field-mapping>  
      <field-name>custNo</field-name>  
      <column-name>CUST_NO</column-name>  
    </cmp-field-mapping>
```



```

    <cmp-field-mapping>
        <field-name>branchCode</field-name>
        <column-name>BRANCH_CODE</column-name>
    </cmp-field-mapping>
    <cmp-field-mapping>
        <field-name>accTypeId</field-name>
        <column-name>ACCTYPE_ID</column-name>
    </cmp-field-mapping>
    <cmp-field-mapping>
        <field-name>accNo</field-name>
        <column-name>ACC_NO</column-name>
    </cmp-field-mapping>
    <cmp-field-mapping>
        <field-name>accBalance</field-name>
        <column-name>ACC_BALANCE</column-name>
    </cmp-field-mapping>
</entity-mapping>
</sun-cmp-mapping>

```

NOTE: The header part of the document, namely the XML declaration and DOCTYPE definition are not included here for brevity.

Entries for all the CMP entity beans have to be made.

Assembling Application for Deployment

Sun Java System Application Server 7 2004Q2 primarily supports the J2EE model wherein applications are packaged in the form of an enterprise archive (EAR) file (extension .ear). The application is further subdivided into a collection of J2EE modules, packaged into Java archives (JAR, extension .jar) for EJBs and web archives (WAR, extension .war) for servlets and JSPs.

All the JSPs and Servlets should be packaged into WAR file, all EJBs into the JAR file and finally the WAR and the JAR file together with the deployment descriptors in to the EAR file. This EAR file is a deployable component.

Deploying iBank application on Sun Java System Application Server 7 2004Q2 using the *asadmin* utility

The last stage is to deploy the application on an instance of Sun Java System Application Server 7 2004Q2. The process for deploying an application is described below:

The Sun Java System Application Server 7 2004Q2 *asadmin* includes a help section on deployment that is accessible from the Help menu.

The command line utility *asadmin* can be invoked by executing *asadmin.bat* file in Windows and *asadmin* file in Solaris Operating Environment that is stored in Sun Java System Application Server 7 2004Q2 installation's bin directory. i.e., *Install_dir/AppServer7/appserv/bin*.

At *asadmin* prompt, the command for deployment would be:

```
asadmin> deploy -u username -w password -H hostname -p adminport [--type
application|ejb|web|client|connector] [--contextroot contextroot]
[--force=true] [--name component-name] [--upload=true] [--instance
instancename] filepath
```

Restart the server instance and then test the application on the browser by typing the url 'http://<machine_name>:<port_number>/iBank'. Test by giving one of the available user name and password. This should show the main menu page of the iBank application.

iBank Application Specification

The *iBank* application is used as the migration sample. This application simulates a basic online banking service with the following functionality:

- log on to the online banking service
- view/edit personal details and branch details
- summary view of accounts showing cleared balances
- facility to drill down by account to view individual transaction history
- money transfer service, allowing online transfer of funds between accounts
- compound interest earnings projection over a number of years for a given principal and annual yield rate

The application is designed after the MVC (Model-View-Controller) model where:

- EJBs are used to define the business and data model components of the application
- Java Server Pages handle the presentation logic and represent the View.
- Servlets play the role of Controllers and handle application logic, taking charge of calling the business logic components and accessing business data via EJBs (the Model), and dispatching processed data for display to Java Server Pages (the View).

Standard J2EE methods are used for assembling and deploying the application components. This includes the definition of deployment descriptors and assembling the application components within the archive files:

- a WAR archive file for the Web application including HTML pages, images, Servlets, JSPs and custom tag libraries, and ancillary server-side Java classes.

- EJB-JAR archive files for the assembling of one or more EJBs, including deployment descriptor, bean class and interfaces, stub and skeleton classes, and other helper classes as required.
- an EAR archive file for the packaging of the enterprise application module that includes the Web application module and the EJB modules used by the application.

The use of standard J2EE assembling methods will be useful in pointing out any differences between Sun ONE Application Server 6.0/6.5 and Sun Java System Application Server 7 2004Q2, and any issues arising thereof.

Tools used for the development of the application

Sun ONE Studio Enterprise Edition, Release 5.0

Sun Java System Application Server 7 2004Q2 supports both the EJB 1.0 and EJB 1.1 standard, the other EJBs in the iBank application (2 session EJBs and the BMP entity bean) that are developed with Sun ONE Studio for Java, and then packaged and deployed in Sun Java System Application Server 7 2004Q2 using the `asadmin deploy` command. This approach also helps in testing the usage of a third-party IDE for developing 1.1 EJBs in Sun Java System Application Server 7 2004Q2 in addition to demonstrating the migration of 1.1 EJBs developed for Sun ONE Application Server 6.5 to Sun Java System Application Server 7 2004Q2.

The Sun ONE Studio development environment is also used to migrate EJB components in the iBank application to Sun Java System Application Server (code adapted from EJB 1.0 standard to EJB 1.1, O/R mapping for CMP entity beans, configuration of deployment properties and packaging of the application's different modules).

Oracle 8i 8.1.6

The database was developed with Oracle 8i (version 8.1.6) and the JDBC driver used to access the database was the thin Oracle driver (type 4).

Database schema

The iBank database schema is derived from the following business rules:

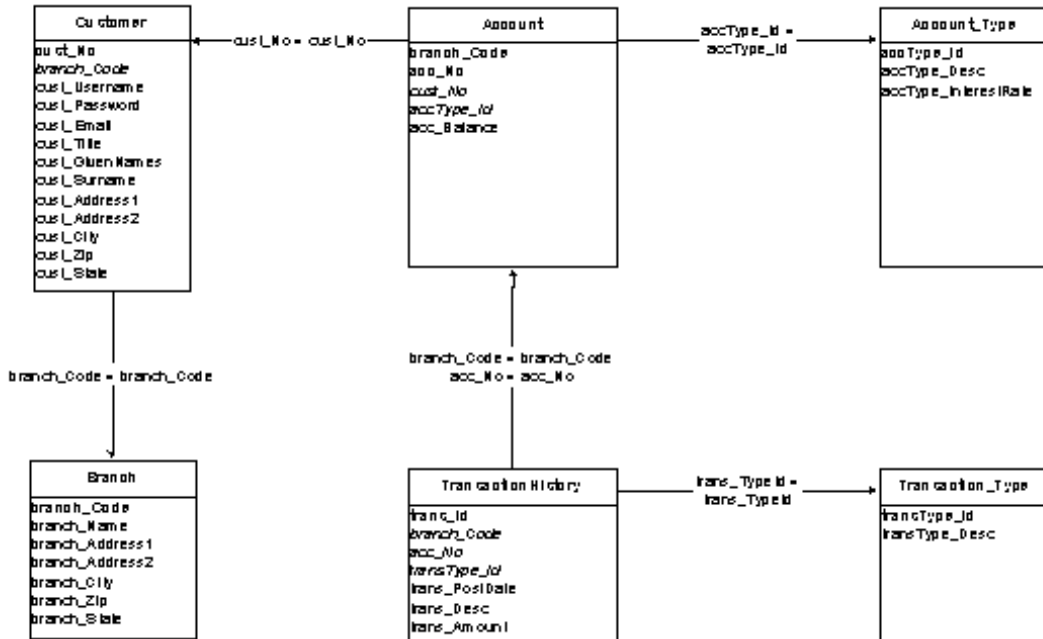
- The iBank company has local branches in major cities.
- A Branch manages all customers within its regional area.
- A Customer has one or more accounts held at their regional branch.

- A customer Account is uniquely identified by the branch code and account number, and also holds the number of the customer to which it belongs. The current cleared balance available is also stored with the account.
- Accounts are of a particular Account Type that is used to distinguish between several kinds of accounts (checking account, savings account, etc.).
- Each Account Type stores a number of particulars that apply to all accounts of this type (regardless of branch or customer) such as interest rate and allowed overdraft limit.
- Every time a customer receives or pays money into/from one of their accounts, the transaction is recorded in a global transaction log, the Transaction History.
- The Transaction History stores details about individual transactions, such as the relevant branch code and account number, the date the transaction was posted (recorded), a code identifying the type of transaction and a complementary description of the particular transaction, and the amount for the transaction.
- Transaction types allow different types of transactions to be distinguished, such as cash deposit, credit card payment, fund transfer between accounts, and so on.

These business rules are illustrated in the entity-relationship diagram below:

Figure A-1 Database Schema

TMBank -- Database schema



The database model translates as a series of table definitions below, where primary key columns are printed in bold type, while foreign key columns are shown in italics.

BRANCH			
BRANCH_CODE	CHAR(4)	NOT NULL	4-digit code identifying the branch
BRANCH_NAME	VARCHAR(40)	NOT NULL	Name of the branch
BRANCH_ADDRESS1	VARCHAR(60)	NOT NULL	Branch postal address, street address, 1st line
BRANCH_ADDRESS2	VARCHAR(60)		Branch postal address, street address, 2nd line
BRANCH_CITY	VARCHAR(30)	NOT NULL	Branch postal address, City

BRANCH_ZIP	VARCHAR(10)	NOT NULL	Branch postal address, Zip code
BRANCH_STATE	CHAR(2)	NOT NULL	Branch postal address, State abbreviation

CUSTOMER			
CUST_NO	INT	NOT NULL	iBank customer number (global)
BRANCH_CODE	CHAR(4)	NOT NULL	References this customer's branch
CUST_USERNAME	VARCHAR(16)	NOT NULL	Customer's login username
CUST_PASSWORD	VARCHAR(10)	NOT NULL	Customer's login password
CUST_EMAIL	VARCHAR(40)		Customer's e-mail address
CUST_TITLE	VARCHAR(3)	NOT NULL	Customer's courtesy title
CUST_GIVENNAMES	VARCHAR(40)	NOT NULL	Customer's given names
CUST_SURNAME	VARCHAR(40)	NOT NULL	Customer's family name
CUST_ADDRESS1	VARCHAR(60)	NOT NULL	Customer postal address, street address, 1st line
CUST_ADDRESS2	VARCHAR(60)		Customer postal address, street address, 2nd line
CUST_CITY	VARCHAR(30)	NOT NULL	Customer postal address, City
CUST_ZIP	VARCHAR(10)	NOT NULL	Customer postal address, Zip code
CUST_STATE	CHAR(2)	NOT NULL	Customer postal address, State abbreviation

ACCOUNT_TYPE			
ACCTYPE_ID	CHAR(3)	NOT NULL	3-letter account type code
ACCTYPE_DESC	VARCHAR(30)	NOT NULL	Account type description
ACCTYPE_INTERESTRATE	DECIMAL(4,2)	DEFAULT 0.0	Annual interest rate

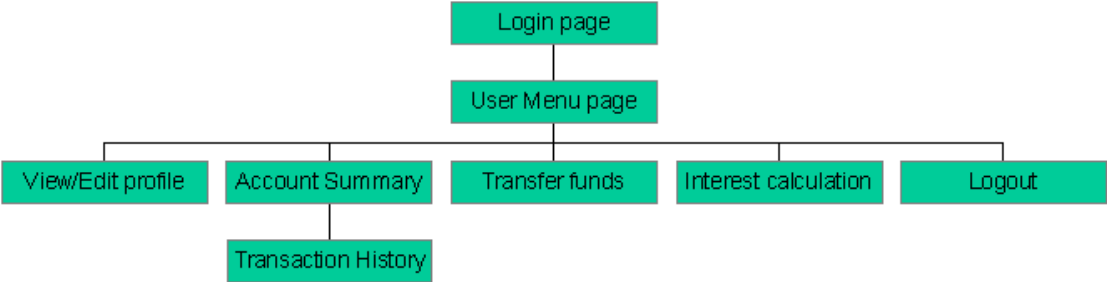
ACCOUNT			
BRANCH_CODE	CHAR(4)	NOT NULL	branch code (primary-key part 1)
ACC_NO	CHAR(8)	NOT NULL	account no. (primary-key part 2)
CUST_NO	INT	NOT NULL	Customer to whom accounts belongs
ACCTYPE_ID	CHAR(3)	NOT NULL	Account type, references ACCOUNT_TYPE

ACC_BALANCE	DECIMAL(10,2)	DEFAULT 0.0	Cleared balance available
TRANSACTION_TYPE			
TRANSTYPE_ID	CHAR(4)	NOT NULL	A 4-letter transaction type code
TRANSTYPE_DESC	VARCHAR(40)	NOT NULL	Human-readable description of code
TRANSACTION_HISTORY			
TRANS_ID	LONGINT	NOT NULL	Global transaction serial no
BRANCH_CODE	CHAR(4)	NOT NULL	key referencing ACCOUNT part 1
ACC_NO	CHAR(8)	NOT NULL	key referencing ACCOUNT part 2
TRANSTYPE_ID	CHAR(4)	NOT NULL	References TRANSACTION_TYPE
TRANS_POSTDATE	TIMESTAMP	NOT NULL	Date & time transaction was posted
TRANS_DESC	VARCHAR(40)		Additional details for the transaction
TRANS_AMOUNT	DECIMAL(10,2)	NOT NULL	Money amount for this transaction

Application navigation and logic

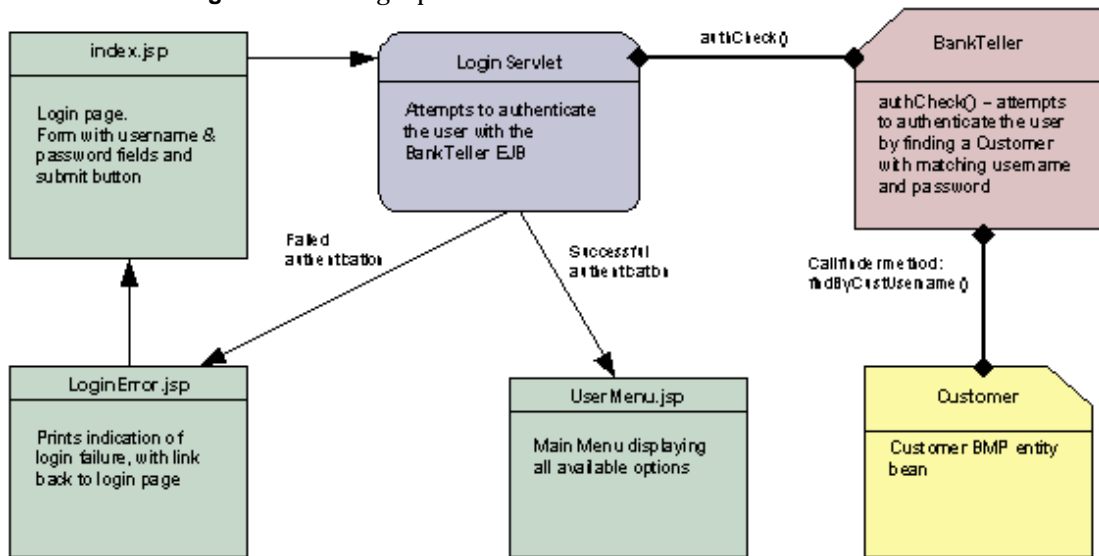
High-level view of application navigation

Figure A-2 Application navigation and logic



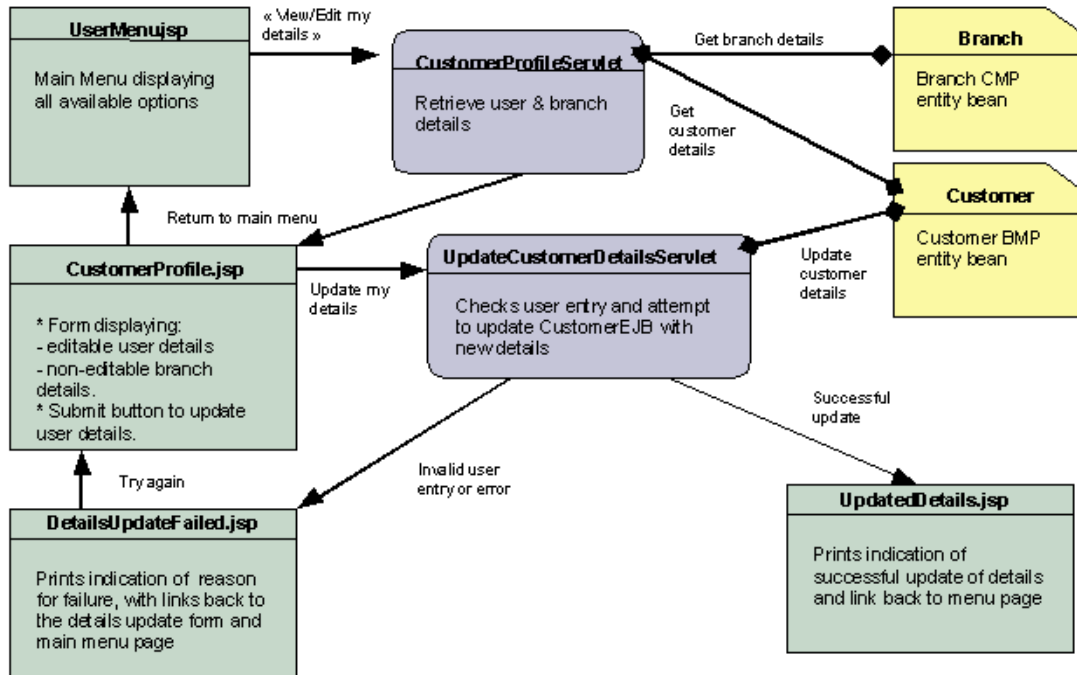
- Login Process

Figure A-3 Login process



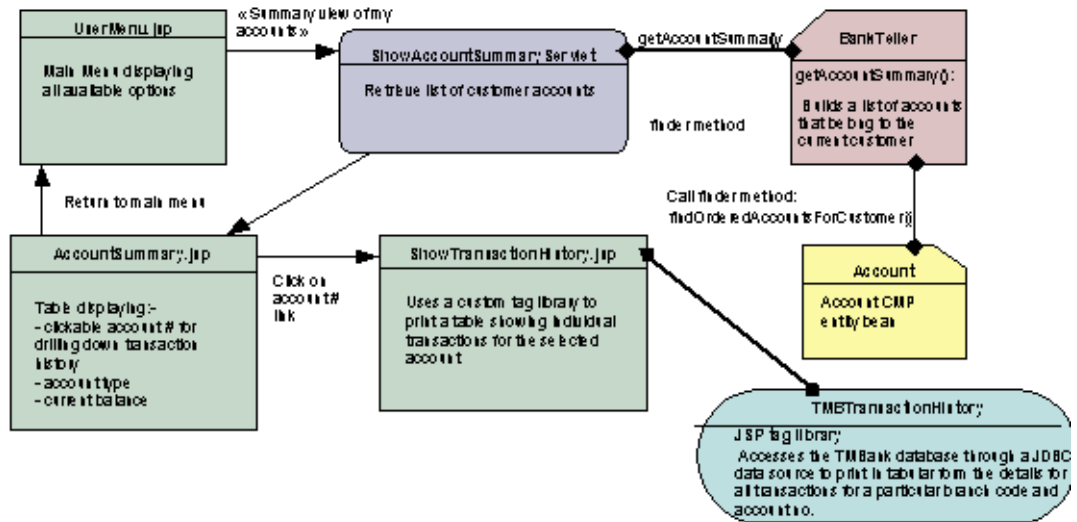
- View / edit details

Figure A-4 View/edit details process



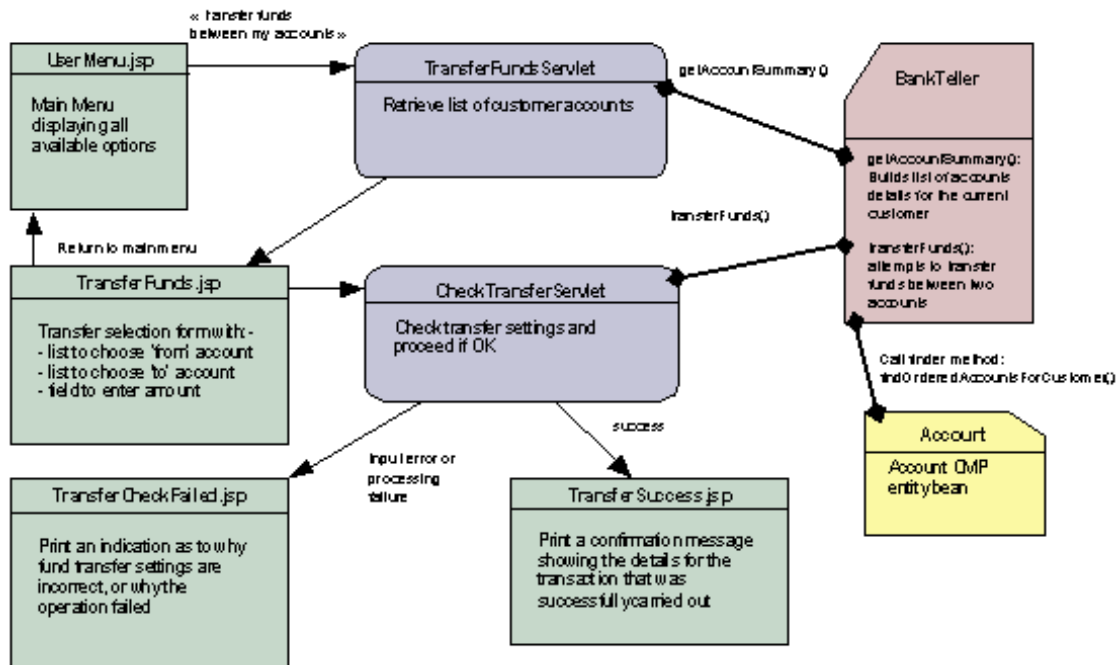
- Account summary and Transaction history

Figure A-5 Account summary and transaction history



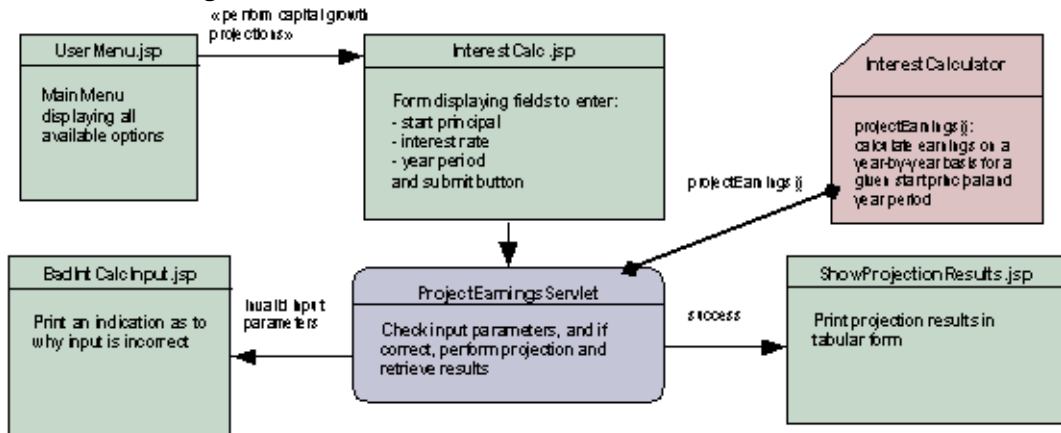
- Fund Transfer

Figure A-6 Fund transfer



- Interest Calculation

Figure A-7 Interest calculation



Application Components

- Data Components

Each table in the database schema is encapsulated as an entity bean:

Entity Bean	Database Table
Account	ACCOUNT table
AccountType	ACCOUNT_TYPE table
Branch	BRANCH table
Customer	CUSTOMER table
Transaction	TRANSACTION_HISTORY table
TransactionType	TRANSACTION_TYPE table

All entity beans use container-managed persistence (CMP), except **Customer**, which uses bean-managed persistence (BMP).

Currently, the application only makes use of the **Account**, **AccountType**, **Branch**, and **Customer** beans.

- Business components

Business components of the application are encapsulated by session beans-

The `BankTeller` bean is a stateful session bean that encapsulates all interaction between the customer and the system. `BankTeller` is notably in charge of the following activities:

- authenticating a customer through the `authCheck()` method
- giving the list of accounts for the customer through the `getAccountSummary()` method
- transferring funds between accounts on behalf of the customer through the `transferFunds()` method

The `InterestCalculator` bean is a stateless session bean that encapsulates financial calculations. It is responsible for providing the compound interest projection calculations, through the `projectEarnings()` method.

- Application logic components (servlets)

Component name	Purpose
LoginServlet	Authenticates the user with the <code>BankTeller</code> session bean (<code>authCheck()</code> method), creates the HTTP session and saves information pertaining to the user in the session. Upon successful authentication, forwards request to the main menu page (<code>UserMenu.jsp</code>)
CustomerProfileServlet	Retrieves customer and branch details from the Customer and Branch entity beans and forwards request to the view/edit details page (<code>CustomerProfile.jsp</code>).
UpdateCustomerDetailsServlet	Attempts to effect customer details changes amended in <code>CustomerProfile.jsp</code> by updating the Customer entity bean after checking validity of changes. Redirects to <code>UpdatedDetails.jsp</code> if success, or to <code>DetailsUpdateFailed.jsp</code> in case of incorrect input.
ShowAccountSummaryServlet	Retrieves the list of customer accounts from the <code>BankTeller</code> session bean (<code>getAccountSummary()</code> method) and forwards request to <code>AccountSummary.jsp</code> for display.
TransferFundsServlet	Retrieves the list of customer accounts from the <code>BankTeller</code> session bean (<code>getAccountSummary()</code> method) and forwards request to <code>TransferFunds.jsp</code> allowing the user to set up the transfer operation.
CheckTransferServlet	Checks the validity of source and destination accounts selected by the user for transfer and the amount entered. Calls the <code>transferFunds()</code> method of the <code>BankTeller</code> session bean to perform the transfer operation. Redirects the user to <code>CheckTransferFailed.jsp</code> in case of input error or processing error, or to <code>TransferSuccess.jsp</code> if the operation was successfully carried out.

ProjectEarningsServlet	Retrieves the interest calculation parameters defined by the user in InterestCalc.jsp and calls the <code>projectEarnings()</code> method of the InterestCalculator stateless session bean to perform the calculation, and forwards results to the <code>ShowProjectionResults.jsp</code> page for display. In case of invalid input, redirects to <code>BadIntCalcInput.jsp</code>
------------------------	---

- **Presentation logic components (JSP Pages)**

Component name	Purpose
<code>index.jsp</code>	Index page to the application that also serves as the login page.
<code>LoginError.jsp</code>	Login error page displayed in case of invalid user credentials supplied. Prints an indication as to why login was unsuccessful.
<code>Header.jsp</code>	Page header that is dynamically included in every HTML page of the application
<code>CheckSession.jsp</code>	This page is statically included in every page in the application and serves to verify whether the user is logged in (i.e. has a valid HTTP session). If no valid session is active, the user is redirected to the <code>NotLoggedIn.jsp</code> page.
NotLoggedIn.jsp	Page that the user gets redirected to when they try to access an application page without having gone through the login process first.
UserMenu.jsp	Main application menu page that the user gets redirected to after successfully logging in. This page provides links to all available actions.
<code>CustomerProfile.jsp</code>	Page displaying editable customer details and static branch details. This page allows the customer to amend their correspondence address.
<code>UpdatedDetails.jsp</code>	Page where the user gets redirected to after successfully updating their details.
<code>DetailsUpdateFailed.jsp</code>	Page where the user gets redirected if an input error prevents their details to be updated.
<code>AccountSummaryPage.jsp</code>	This page displays the list of accounts belonging to the customer in tabular form listing the account no, account type and current balance. Clicking on an account no. in the table causes the application to present a detailed transaction history for the selected account.
<code>ShowTransactionHistory.jsp</code>	This page prints the detailed transaction history for a particular account no. The transaction history is printed using a custom tag library.
<code>TransferFunds.jsp</code>	This page allows the user to set up a transfer from one account to another for a specific amount of money.
<code>TransferCheckFailed.jsp</code>	When the user chooses incorrect settings for fund transfer, they get redirected to this page.
<code>TransferSuccess.jsp</code>	When the fund transfer set-up by the user can successfully be carried out, this page will be displayed, showing a confirmation message.

InterestCalc.jsp	This page allows the user to enter parameters for a compound interest calculation.
BadIntCalcInput.jsp	If the parameters for compound interest calculation are incorrect, the user gets redirected to this page.
ShowProjectionResults.jsp	When an interest calculation is successfully carried out, the user is redirected to this page that displays the projection results in tabular form.
Logout.jsp	Exit page of the application. This page removes the stateful session bean associated with the user and invalidates the HTTP session.
Error.jsp	In case of unexpected application error, the user will be redirected to this page that will print details about the exception that occurred.

Fitness of design choices with regard to potential migration issues

While many of application design choices made are certainly debatable especially in the "real-world" context, care was taken to ensure that these choices enable the sample application to encompass as many potential issues as possible as one would face in the process of migrating a typical J2EE application.

This section will go through the potential issues that you might face when migrating a J2EE application, and the corresponding component of iBank that was included to check for this issue during the migration process.

With respect to the selected migration areas to address, this section specifically looks at the following technologies:

Servlets

The iBank application includes a number of servlets, that enable us to detect potential issues with:

- The use of generic functionality of the Servlet API.
- Storage/retrieval of attributes in the HTTP session and HTTP request.
- Retrieval of servlet context initialization parameters.
- Page redirection.

Java Server Pages

With respect to the JSP specification, the following aspects have been addressed:

- Use of JSP declarations, scriptlets, expressions, and comments.
- Static includes (`<%@ include file="..." %>`): notably tested with the inclusion of the `CheckSession.jsp` file in every page).

- Dynamic includes (`<jsp:include page=... />`): this is catered for by the dynamic inclusion of `Header.jsp` in every page.
- Use of custom tag libraries: a custom tag library is used in `ShowTransactionHistory.jsp`.
- Error pages for JSP exception handling: the `Error.jsp` page is the application error redirection page.

JDBC

The iBank application accesses a database via a connection pool and the data source, both programmatically (BMP entity bean, BankTeller session bean, custom tag library) and declaratively (with the CMP entity beans).

Enterprise Java Beans

The iBank application uses a variety of Enterprise Java Beans:

Entity beans:

Bean-managed persistence (`Customer` bean): allows us to test the following:

- JNDI lookup of initial context
- pooled data source access via JDBC
- definition of a BMP custom finder (`findByCustUsername()`)

Container-managed persistence ("`Account`" and "`Branch`" beans): allow us to test the following:

- Object/Relational mapping with the development tool and within the deployment descriptor
- Use of composite primary keys (`Account`)
- Definition of custom CMP finders (with the "`Account`" bean, and its `findOrderedAccountsForCustomer()` method). This is the occasion to look at differences in declaring the query logic in the deployment descriptor, and also to have a complex example returning a collection of objects.

Session beans:

Stateless session beans: `InterestCalculator` allows us to test the following:

- using and deploying a stateless session bean
- calling a business method for calculations

Stateful session beans: `BankTeller` allows us to test the following:

- looking up various interfaces using JNDI and initial contexts
- using JDBC to perform database queries
- using various transactional attributes on bean methods
- using container-demarcated transactions
- maintaining conversational state between calls
- business methods acting as front-ends to entity beans (e.g., the `"getAccountSummary()"` method)

Application Assembly

iBank is assembled by following the J2EE standard procedures. It contains the following components:

- a Web application archive file for the Web application module, and EJB-JAR archives for the EJBs.
- an Enterprise application archive file (EAR file) for the final packaging of the Web application and EJB modules.

Migration Resources

This appendix briefly describes the resources that helps in the migration of J2EE application components to Sun Java System Application Server 7 2004Q2 Standard and Enterprise Edition.

This appendix contains the following topics:

- [Migrating Applications From Competitive Application Servers](#)
- [Migration Tools](#)
- [References](#)
- [Redeploying Migrated Applications](#)

Migrating Applications From Competitive Application Servers

Applications built with J2EE technology can be moved from one application server to another without much modification to the source code. This section gives information about the resources that helps you migrate applications built on competitive application servers to Sun Java System Application Server 7 2004Q2.

Migrating from BEA WebLogic to Sun Java System Application Server 7

To migrate your applications built on J2EE technology from BEA Web Logic Application Server 5.1 or 6.x to Sun Java System Application Server 7 2004Q2, refer to the guides located at the following URL:

http://iforce.sun.com/partners/migration/pdf/mgr_resource_guide_0204.pdf

These guides provide an overview of the migration strategy, migration process, migration considerations and also the actual migration of a sample application.

Migrating from IBM WebSphere to Sun Java System Application Server 7

To migrate your applications built on J2EE technology from IBM Web Sphere Application Server to Sun ONE Application Server 7, refer to the instructions provided in the guide at the following URL:

http://iforce.sun.com/partners/migration/pdf/mgr_resource_guide_0204.pdf

This guide provides an understanding of the target and source application server in terms of architecture and features. This guide also discusses about the migration issues and actual migration of a sample application.

Migration Tools

Sun Java System Application Server supports various migration tools that automates the migration of various J2EE application components. This section provides resource information on the supported tools.

Sun ONE Studio, Release 5 Update 1

The Sun ONE Studio for Java Development Environment is used to migrate various components of J2EE application to Sun Java System Application Server 7 2004Q2.

For general information about the tool and its features, see Sun ONE Studio for Java documentation at the following URL:

<http://docs.sun.com/db/prod/7186#hic>

Sun Java System Migration Tool for Application Server

Sun Java System Migration Tool 3.x/4.x for Application Servers automates the migration of J2EE applications to Sun ONE Application Server 7 and Sun Java System Application Server 7 2004Q2, without much modification to the source code.

The key features of the tool are:

- Migration of application server-specific deployment descriptors
- Runtime support for selected custom JavaServer Pages™ (JSP™) tags and proprietary APIs
- Conversion of selected configuration parameters with equivalent functionality in Sun Java System Application Server
- Automatic generation of Ant based scripts for building and deploying the migrated application to the target Application Server
- Generation of comprehensive migration reports after achieving migration

You can download the tool from the following location:

http://www.sun.com/software/download/app_servers.html

For detailed information on how to install and use the tool, consult its online help.

Sun ONE Migration Toolbox for Applogic and NetDynamics

Sun ONE Migration Toolbox is used to migrate applications built on NetDynamics or Kiva/NAS platforms to Sun ONE Application Server 6.5. Next, use the Migration Tool to migrate 6.5 applications to Sun Java System Application Server 7 2004Q2. The main interface for the Sun ONE Migration Toolbox is the *Toolbox application*. This application can be invoked by running the `%MIGTBX_HOME%/bin/toolbox.bat` script (provided the `setenv.bat` file has been customized appropriately, see `README.txt` for more information).

The following source platforms are supported for migration using the Toolbox:

- Windows NT 4.0
- Windows 2000

Although it is expected that the application can be run on other Windows platforms (Windows 95/98/Me), these platforms have not been tested and may require additional configuration beyond that is specified in the Sun ONE Migration Toolbox installation documentation.

The Toolbox requires the JDK 1.2.2 (JDK 1.3.1 has been tested) to run successfully.

J2EE Application Verification Kit

The Java Application Verification Kit (AVK) for the Enterprise helps you build and test your applications for correct use of J2EE APIs and migrate to other J2EE compatible application servers using specific guidelines and rules.

You can download the Java Application Verification Kit (AVK) from the following location:

<http://java.sun.com/j2ee/verified/>

References

This section gives you references to the Migration Documents of Sun ONE Application Server 6.x.

Migrating to Sun ONE Application Server 6.0

For information about migrating your KIVA/NAS/NetDynamics applications to Sun ONE Application Server 6.0, see the *Sun ONE Application Server Migration Guide* at the following URL:

<http://docs.sun.com/db/doc/816-5780-10>

Migrating to Sun ONE Application Server 6.5

For information about migrating your KIVA/NAS/NetDynamics applications to Sun ONE Application Server 6.5, see the *Sun ONE Application Server 6.5 Migration Guide* at the following URL:

<http://docs.sun.com/db/doc/816-5793-11>

Migrating to Sun ONE Application Server 7

For information about migrating your Sun ONE Application Server 6.x applications to Sun ONE Application Server 7, see *Sun ONE Application Server 7 Migrating and Redeploying Server Applications Guide* at the following URL:

<http://docs.sun.com/db/doc/817-2158-10>

Redeploying Migrated Applications

Most of the applications that are migrated automatically through the use of the available migration tools will utilize the standard deployment tasks described in the *Sun Java System Application Server Administration Guide*.

In some cases, the automatic migration will not be able to migrate particular methods or syntaxes from the source application. When this occurs in the case of the Sun Java System Migration Tool for Application Servers, you are notified of the steps that will be needed to complete the migration. Once you complete the post-migration manual steps, you will be able to deploy the application in the standard manner described in the *Sun Java System Application Server Administration Guide*.

Migrating from the Enterprise Java Beans 1.1 Specification to Enterprise Java Beans 2.0

Although the Enterprise Java Beans™ (EJB) 1.1 specification continues to be supported in Sun Java System Application Server 7 2004Q2, the use of the EJB 2.0 architecture is recommended to leverage its enhanced capabilities. To migrate EJB 1.1 to EJB 2.0 a number of modifications will be required, including within the source code of components.

This chapter contains the following sections:

- [Differences Between EJB 1.1 and EJB 2.0](#)
- [Migrating EJB Client Applications](#)
- [Migrating CMP Entity EJBs](#)
- [Migrating the Bean Class](#)
- [Migrating ejb-jar.xml](#)
- [Custom Finder Methods](#)

Differences Between EJB 1.1 and EJB 2.0

Essentially, the required modifications relate to the differences between EJB 1.1 and EJB 2.0, all of which are described in the following topics:

- [EJB Query Language](#)
- [Local Interfaces](#)

- [EJB 2.0 Container-Managed Persistence \(CMP\)](#)
- [Defining Persistent Fields](#)
- [Defining Entity Bean Relationships](#)
- [Message-Driven Beans](#)

EJB Query Language

The EJB 1.1 specification left the manner and language for forming and expressing queries for finder methods to each individual application server. While many application server vendors let developers form queries using SQL, others use their own proprietary language specific to their particular application server product. This mixture of query implementations causes inconsistencies between application servers.

The EJB 2.0 specification introduces a query language called *EJB Query Language*, or *EJB QL* to correct many of these inconsistencies and shortcomings. EJB QL is based on SQL92. It defines query methods, in the form of both finder and select methods, specifically for entity beans with container-managed persistence. EJB QL's principal advantage over SQL is its portability across EJB containers and its ability to navigate entity bean relationships.

Local Interfaces

In the EJB 1.1 architecture, session and entity beans have one type of interface, a remote interface, through which they can be accessed by clients and other application components. The remote interface is designed such that a bean instance has remote capabilities; the bean inherits from RMI and can interact with distributed clients across the network.

With EJB 2.0, session beans and entity beans can expose their methods to clients through two types of interfaces: a *remote interface* and a *local interface*. The 2.0 remote interface is identical to the remote interface used in the 1.1 architecture, whereby, the bean inherits from RMI, exposes its methods across the network tier, and has the same capability to interact with distributed clients.

However, the local interfaces for session and entity beans provide support for lightweight access from EJBs that are local clients; that is, clients co-located in the same EJB container. The EJB 2.0 specification further requires that EJBs that use local interfaces be within the same application. That is, the deployment descriptors for an application's EJBs using local interfaces must be contained within one `ejb-jar` file.

The local interface is a standard Java interface. It does not inherit from `RMI`. An enterprise bean uses the local interface to expose its methods to other beans that reside within the same container. By using a local interface, a bean may be more tightly coupled with its clients and may be directly accessed without the overhead of a remote method call.

In addition, local interfaces permit values to be passed between beans with pass by reference semantics. Because you are now passing a reference to an object, rather than the object itself, this reduces the overhead incurred when passing objects with large amounts of data, resulting in a performance gain.

Setting up a session or entity bean to use a local interface rather than a remote interface is simple. The local interface through which the bean's methods are exposed to clients extends `EJBLocalObject` rather than `EJBObject`. Similarly, the bean's home interface extends `EJBLocalHome` rather than `EJBHome`. The implementation class extends the same `EntityBean` or `SessionBean` interface.

NOTE A bean destined to be remote in EJB 2.0 extends `EJBObject` in its remote interface and `EJBHome` in its home interface, just as it did in EJB 1.1.

EJB 2.0 Container-Managed Persistence (CMP)

The EJB 2.0 specification has expanded CMP to allow multiple entity beans to have relationships among themselves. This is referred to as *Container-Managed Relationships* (CMR). The container manages the relationships and the referential integrity of the relationships.

The EJB 1.1 specification presented a more limited CMP model. The 1.1 architecture limited CMP to data access that is independent of the database or resource manager type. It allowed you to expose only an entity bean's instance state through its remote interface; there is no means to expose bean relationships. The 1.1 version of CMP depends on mapping the instance variables of an entity bean class to the

data items representing their state in the database or resource manager. The CMP instance fields are specified in the deployment descriptor, and when the bean is deployed, the deployer uses tools to generate code that implements the mapping of the instance fields to the data items.

You must also change the way you code the bean's implementation class. According to the 2.0 specification, the implementation class for an entity bean that uses CMP is now defined as an abstract class.

Defining Persistent Fields

The EJB 2.0 specification lets you designate an entity bean's instance variables as CMP fields or CMR fields. You define these fields in the deployment descriptor. CMP fields are marked with the element `cmp-field`, while container-managed relationship fields are marked with the element `cmr-field`.

In the implementation class, note that you do not declare the CMP and CMR fields as public variables. Instead, you define `get` and `set` methods in the entity bean to retrieve and set the values of these CMP and CMR fields. In this sense, beans using the 2.0 CMP follow the JavaBeans model: instead of accessing instance variables directly, clients use the entity bean's `get` and `set` methods to retrieve and set these instance variables. Keep in mind that the `get` and `set` methods only pertain to variables that have been designated as CMP or CMR fields.

Defining Entity Bean Relationships

As noted previously, the EJB 1.1 architecture does not support CMRs between entity beans. The EJB 2.0 architecture does support both one-to-one and one-to-many CMRs. Relationships are expressed using CMR fields, and these fields are marked as such in the deployment descriptor. You set up the CMR fields in the deployment descriptor using the appropriate deployment tool for your application server.

Similar to CMP fields, the bean does not declare the CMR fields as instance variables. Instead, the bean provides `get` and `set` methods for these fields.

Message-Driven Beans

Message-driven beans are another new feature introduced by the EJB 2.0 architecture. Message-driven beans are transaction-aware components that process asynchronous messages delivered through the Java Message Service (JMS). The JMS API is an integral part of the J2EE 1.3 platform.

Asynchronous messaging allows applications to communicate by exchanging messages so that senders are independent of receivers. The sender sends its message and does not have to wait for the receiver to receive or process that message. This differs from synchronous communication, which requires the component that is invoking a method on another component to wait or block until the processing completes and control returns to the caller component.

Migrating EJB Client Applications

This section includes the following topics:

- [Declaring EJBs in the JNDI Context](#)
- [Recap on Using EJB JNDI References](#)

Declaring EJBs in the JNDI Context

In Sun Java System Application Server 7 2004Q2, EJBs are systematically mapped to the JNDI sub-context "*ejb/*". If we attribute the JNDI name "*Account*" to an EJB, then Sun Java System Application Server 7 2004Q2 will automatically create the reference "*ejb/Account*" in the global JNDI context. The clients of this EJB will therefore have to look up "*ejb/Account*" to retrieve the corresponding home interface.

Let us examine the code for a servlet method deployed in Sun ONE Application Server 6.0/6.5,

The servlet presented here calls on a stateful session bean, *BankTeller*, mapped to the root of the JNDI context. The method whose code we are considering is responsible for retrieving the home interface of the EJB, so as to enable a *BankTeller* object to be instantiated and a remote interface for this object to be retrieved, in order to make business method calls to this component.

```
/**
```

```
 * Look up the BankTellerHome interface using JNDI.
```

```

        */

private BankTellerHome lookupBankTellerHome(Context ctx)
    throws NamingException
{
    try
    {
        Object home = (BankTellerHome) ctx.lookup("ejb/BankTeller");
        return (BankTellerHome) PortableRemoteObject.narrow(home,
BankTellerHome.class);
    }
    catch (NamingException ne)
    {
        log("lookupBankTellerHome: unable to lookup BankTellerHome" +
            "with JNDI name 'BankTeller': " + ne.getMessage() );
        throw ne;
    }
}

```

As the code already uses `ejb/BankTeller` as an argument for the lookup, there is no need for modifying the code to be deployed on Sun Java System Application Server 7 2004Q2.

Recap on Using EJB JNDI References

This section summarizes the considerations when using EJB JNDI references. Where noted, the consideration details are specific to a particular source application server platform.

Placing EJB References in the JNDI Context

It is only necessary to modify the name of the EJB references in the JNDI context mentioned above (moving these references from the JNDI context root to the sub-context "*ejb*") when the EJBs are mapped to the root of the JNDI context in the existing WebLogic application.

If these EJBs are already mapped to the JNDI sub-context `ejb/` in the existing application, no modification is required.

However, when configuring the JNDI names of EJBs in the deployment descriptor within the Sun ONE Studio for Java, it is important to avoid including the prefix `ejb/` in the JNDI name of an EJB. Remember that these EJB references are *automatically* placed in the JNDI `ejb/` sub-context with Sun Java System Application Server 7 2004Q2. So, if an EJB is given to the JNDI name "*BankTeller*" in its deployment descriptor, the reference to this EJB will be "translated" by Sun Java System Application Server into `ejb/BankTeller`, and this is the JNDI name that client components of this EJB must use when carrying out a lookup.

Global JNDI context versus local JNDI context

Using the global JNDI context to obtain EJB references is a perfectly valid, feasible approach with Sun Java System Application Server 7 2004Q2. Nonetheless, it is preferable to stay as close as possible to the J2EE specification, and retrieve EJB references through the local JNDI context of EJB client applications. When using the local JNDI context, you must first declare EJB resource references in the deployment descriptor of the client part (`web.xml` for a Web application, `ejb-jar.xml` for an EJB component).

Migrating CMP Entity EJBs

This section describes the steps to migrate your application components from the EJB 1.1 architecture to the EJB 2.0 architecture.

In order to migrate a CMP 1.1 bean to CMP 2.0, we first need to verify if a particular bean can be migrated. The steps to perform this verification are as follows.

1. From the `ejb-jar.xml` file, go to the `<cmp-fields>` names and check if the optional tag `<prim-key-field>` is present in the `ejb-jar.xml` and has an indicated value, if yes, go to next step.

Look for the `<prim-key-class>` field name in the `ejb-jar.xml`, get the class name and get the public instance variables declared in the class. Now

2. if the signature (name and case) of these variables matches with the `<cmp-field>` names above. Segregate the ones that are found. In these segregated fields, check if some of them start with an upper case letter. If any of them do, then migration cannot be performed.

3. Look into the bean class source code and obtain the java types of all the <cmp-field> variables.
4. Change all the <cmp-field> names to lowercase and construct accessors from them. For example if the original field name is `Name` and its java type is `String`, the accessor method signature will be:


```
Public void setName(String name)

Public String getName()
```
5. Compare these accessor method signatures with the method signatures in the bean class. If there is an exact match found, migration is not possible.
6. Get the custom finder methods signatures and their corresponding SQLs. Check if there is a 'Join' or 'Outer join' or an 'OrderBy' in the SQL, if yes, we cannot migrate, as EJB QL does not support 'joins', 'Outer join' and 'OrderBy'.
7. Any CMP 1.1 finder, which used `java.util.Enumeration`, should now use `java.util.Collection`. Change your code to reflect this. CMP2.0 finders cannot return `java.util.Enumeration`.

The next topic, *Migrating the Bean Class*, describes the migration process.

Migrating the Bean Class

This section describes the steps required to migrate the bean class to Sun Java System Application Server.

1. Prepend the bean class declaration with the keyword *abstract*. For example if the bean class declaration was:


```
Public class CabinBean implements EntityBean // before modification

abstract Public class CabinBean implements EntityBean // after
modification
```
2. Prefix the accessors with the keyword *abstract*.
3. Insert all the accessors after modification into the source(.java) file of the bean class at class level.
4. Comment out all the `cmp` fields in the source file of the bean class.
5. Construct protected instance variable declarations from the `cmp-field` names in lowercase and insert them at the class level.

6. Read up all the `ejbCreate()` method bodies (there could be more than one `ejbCreate`). Look for the pattern '`<cmp-field>=some value or local variable`', and replace it with the expression 'abstract mutator method name (*same value or local variable*)'. For example, if the `ejbCreate` body (before migration) is like this:

```
public MyPK ejbCreate(int id, String name)
{
    this.id = 10*id;
    Name = name;//1
    return null;
}
```

The changed method body (after migration) should be:

```
public MyPK ejbCreate(int id, String name)
{
    setId(10*id);
    setName(name);//1
    return null;
}
```

NOTE The method signature of the abstract accessor in `//1` is as per the Camel Case convention mandated by the EJB 2.0 spec. Also, the keyword '*this*' may or may not be present in the original source, *but it has to be removed* from the modified source file.

7. All the protected variables declared in the `ejbPostCreate()` methods in Step 5 have to be initialized. The protected variables will be equal in number with the `ejbCreate()` methods. This initialization will be done by inserting the initialization code in the following manner:

```
protected String name;//from step 5
protected int id;//from step 5
public void ejbPostCreate(int id, String name)
```

```

        {
            name /*protected variable*/ = getName();/*abstract accessor*/
            //inserted in this step

            id /*protected variable*/ = getId();/*abstract accessor*/ //inserted in
            this step
        }

```

8. Inside the `ejbLoad` method, you have to set the protected variables to the beans database state. So insert the following lines of code:

```

public void ejbLoad()
{
    name = getName();//inserted in this step
    id = getId(); //inserted in this step
    ..... //already present code
}

```

9. Similarly, you will have to update the beans' state inside `ejbStore()` so that its database state gets updated. But remember, you are not allowed to update the setters that correspond to the primary key outside the `ejbCreate()`, so do not include them inside this method. Insert the following lines of code:

```

public void ejbStore()
{
    setName(name);//inserted in this step
    // setId(id);//Do not insert this if it is a part of the primary
    key
    .....//already present code
}

```

10. As a last change to the bean class source (.java) file, examine the whole code and replace all occurrences of any `<cmp-field>` variable name with the equivalent protected variable name (as declared in Step 5).

If you do not migrate the bean, at the minimum you need to insert the `<cmp-version>1.X</cmp-version>` tag inside the `ejb-jar.xml` at the appropriate place, so that the unmigrated bean still works on Sun Java System Application Server.

Migrating ejb-jar.xml

To migrate the file `ejb-jar.xml` to Sun Java System Application Server perform the following steps:

1. In the `ejb-jar.xml`, convert all `<cmp-fields>` to become lowercase.
2. In the `ejb-jar.xml` file, insert the tag `<abstract-schema-name>` after the `<reentrant>` tag. The schema name will be the name of the bean as in the `<ejb-name>` tag, prefixed with “ias_”.
3. Insert the following tags after the `<primkey-field>` tag:

```
<security-identity><use-caller-identity/></security-identity>
```
4. Use the SQL's obtained above to construct the EJB QL from SQL.
5. Insert the `<query>` tag and all its nested child tags with all the required information in the `ejb-jar.xml`, just after the `<security-identity>` tag.

Custom Finder Methods

The custom finder methods are the `findBy...` methods (other than the default `findByPrimaryKey` method) which can be defined in the home interface of an entity bean. As the EJB 1.1 specification does not stipulate a standard for defining the logic of these finder methods, EJB server vendors are free to choose their implementations. As a result, the procedures used to define the methods vary considerably between the different implementations chosen by vendors.

Sun ONE Application Server 6.0 and 6.5 use standard SQL to specify the finder logic.

Information concerning the definition of this finder method is stored in the EJB's persistence descriptor (`Account-ias-cmp.xml`) as follows:

```
<bean-property>
  <property>
    <name>findOrderedAccountsForCustomerSQL</name>
    <type>java.lang.String</type>
    <value>
      SELECT BRANCH_CODE,ACC_NO FROM ACCOUNT where CUST_NO = ?
    </value>
```

```

        <delimiter>,</delimiter>
    </property>
</bean-property>
<bean-property>
    <property>
        <name>findOrderedAccountsForCustomerParms</name>
        <type>java.lang.Vector</type>
        <value>CustNo</value>
        <delimiter>,</delimiter>
    </property>
</bean-property>

```

Each `findXXX` finder method therefore has two corresponding entries in the deployment descriptor (SQL code for the query, and the associated parameters).

In Sun Java System Application Server the custom finder method logic is also declarative, but is based on the EJB query language EJB QL.

The EJB-QL language cannot be used on its own. It has to be specified inside the file `ejb-jar.xml`, in the `<ejb-ql>` tag. This tag is inside the `<query>` tag, which defines a query (finder or select method) inside an EJB. The EJB container can transform each query into the implementation of the finder or select method. Here's an example of an `<ejb-ql>` tag:

```

<ejb-jar>
  <enterprise-beans>
    <entity>
      <ejb-name>hotelEJB</ejb-name>
      ...
      <abstract-schema-name>TMBankSchemaName</abstract-schema-name>
      <cmp-field>...
      ...
      <query>
        <query-method>
          <method-name>findByCity</method-name>
          <method-params>
            <method-param>java.lang.String</method-param>
          </method-params>
        </query-method>
      </ejb-ql>
    
```

```
        <![CDATA[SELECT OBJECT(t) FROM TMBankSchemaName AS t WHERE t.city  
= ?1]]>  
        </ejb-ql>  
    </query>  
</entity>  
...  
</enterprise-beans>  
...  
</ejb-jar>
```


A

- About Sun Java System Application Server 7 [13](#)
- About Sun ONE Application Server 6.0/6.5 [36](#)
- Administration Server [30](#)
- Administration Tool [29](#)
- Administration Tools [29](#)
 - Sun ONE Application Server 6.0 [29](#)
- application client JAR [33](#)
- Architecture [21, 24](#)
 - Sun Java System Application Server 7 Architecture [22](#)
 - Sun ONE Application Server 6.0/6.5 architecture [36](#)
- asadmin [32, 52, 104](#)

B

- BMP [53](#)

C

- CMP [46, 53](#)

D

- data sources [40](#)
- Database Connectivity [32](#)
- Deployment descriptors [17, 33](#)
- Development Environments [27](#)
 - Sun Java System Application Server 7 [27](#)
 - Sun ONE Application Server 6.0/6.5 [27](#)
- DriverManager [39](#)

E

- EAR files [33](#)

- EJB [46](#)

- EJB 1.1 to EJB 2.0

- Defining Entity Bean Relationships [130](#)

- EJB 2.0 Container-Managed Persistence (CMP) [129](#)

- EJB Query Language [128](#)

- Message-Driven Beans [131](#)

- Migrating CMP Entity EJBs

- Custom Finder Methods [137](#)

- Migrating the Bean Class [134](#)

- Migration of ejb-jar.xml [137](#)

- Migrating EJB Client Applications [131](#)

- Declaring EJBs in the JNDI Context [131](#)

- Migration of ejb-jar.xml [137](#)

- EJB Changes Specific to SISAS 7 [47](#)

- EJB JAR [33](#)

- EJB Migration [46](#)

- EJB QL [46](#)

- Enterprise Applications [57](#)

- Application root context and access URL [58](#)

- Migrating Proprietary Extensions [61](#)

- Enterprise EJB Modules [53](#)

- Enterprise JavaBeans [25](#)

- Entity Beans [49](#)

I

- iBank [37, 62](#)

- iBank Application specification

- Application Components [115](#)

- Application navigation and logic [110](#)

- Database schema [106](#)

- Fitness of design choices with regard to potential migration issues [118](#)

- Tools used for the development of the application [106](#)

J

- J2EE [25](#)

- J2EE Application Components and Migration [32](#)

- J2EE applications
 - components [33](#)
- J2EE Component Standards [25](#)
- JavaServer Pages [25](#)
- JDBC Code [38](#)
 - Using JDBC 2.0 Data Sources [40](#)
 - Configuring a Data Source [40](#)
 - Looking Up the Data Source Via JNDI [43](#)
- JNDI context [43](#)
- JSP 1.2 specification [44](#)
- JSP's and JSP Custom Tag Libraries [44](#)

M

- Manual Migration of iBank Application [82](#)
 - Assembling Application for Deployment [103](#)
 - EJB Changes [83](#)
 - Web application changes [82](#)
- MDB [46](#)
- Migrating [21](#)
- Migrating Applications [35](#)
- Migrating From S1AS 6.x to SJSAS 7 [37](#)
- Migration and Redeployment
 - What is Redeployment [18](#)

O

- Obtaining a Data Source from the JNDI Context [46](#)

R

- Registry Editor [29](#)
- rpm [12](#)

S

- Servlets [25](#), [45](#)
- Session Beans [47](#)
- showrev [12](#)
- Sun customer support [12](#)
- Sun ONE Console [29](#)
- Sun ONE Studio [27](#)

T

- Toolbox application [123](#)

W

- WAR [33](#)
- Web Applications [50](#)
 - Migrating Web Application Modules [51](#)
 - Particular setbacks when migrating servlets and JSPs [52](#)