Oracle® Financial Services

Installation and Configuration Guide

Release 4.5

July 2000

Part No. A80992-01

ORACLE®

Oracle Financial Services Installation and Configuration Guide, Release 4.5

Part No. A80992-01

Primary Authors: Rob Flippo, Steven Roepke

Contributing Authors: Kevin Holliday, Bart Stoehr, Steve Wasserman

Technical Reviewers: Sue Bernstein, Victor Cheung, Lee Coller, Kevin Courtney, Kelley Fon-Ndikum, Greg Hall, John Lightfoot, Rondi Mertes, Sergi Semenov

Contents

5 System Requirements

6 UNIX Server Installation and Configuration

7 Client Software Installation and Configuration

8 Budgeting & Planning Server-Side Installation and Setup

9 Budgeting & Planning Database Upgrade Process

10 FDM Database Installation

11 Upgrading from OFSA 3.5/4.0

12 FDM Database Upgrade Process

15 FDM Multi-Language Support

16 FDM Object Management

17 FDM Leaf Management

18 FDM Database Performance Management

19 OFSA Multiprocessing

20 Request Queue

21 FDM Utilities

Send Us Your Comments

Oracle Financial Services Installation and Configuration Guide, Release 4.5

Part No. A80992-01

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

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

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

- Electronic mail - fsdocmail@us.oracle.com
- FAX - (650) 506-7200 Attn: Oracle Financials Documentation Manager
- Postal service:
  Oracle Corporation
  Oracle Financials Documentation Manager
  500 Oracle Parkway
  Redwood Shores, CA 94065
  USA

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

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

Preface

This reference guide describes the features of the *Oracle Financial Services Installation and Configuration Guide*. This preface describes the following information about the reference guide:

- Intended Audience
- Organization
- Related Documents
- Conventions
- Customer Support Information

Intended Audience

This reference guide is written for Database or System Administrators intending to install or configure any of the applications within the Oracle Financial Services Applications (OFSA) group of applications. It assumes familiarity with database and system administration concepts

Organization

This reference guide is organized into the following chapters:

Chapter 1 Introduces the Oracle Financial Services Applications (OFSA) group of applications. You should read this chapter before using the detailed information in the remainder of the reference guide.

Chapter 2 Describes new features for the OFSA group of applications, as well as new terminology used in Release 4.5.

Chapter 3 Provides details on the platforms, operating systems, and software for which this release of the OFSA group of applications has been certified.

Chapter 4 Describes the architecture employed by the OFSA group of applications.

Chapter 5 Describes the recommended client-side hardware requirements for the OFSA group of applications.

Chapter 6 Provides information on installing the Oracle Financial Services server-centric software and properly configuring your Unix server.

Chapter 7 Provides information on installing, configuring and upgrading the client-side of the Oracle Financial Services Applications group of applications.

Chapter 8 Presents information on the software components required to run the Budgeting & Planning process and additional installation routines essential to completing the Budgeting & Planning installation process.

Chapter 9 Discusses the procedure for upgrading the Budgeting & Planning Express database.

Chapter 10 Provides information on installing the Financial Data Manager (FDM) database.

Chapter 11 Provides information for users upgrading from OFSA database versions 3.5 and 4.0 to the Financial Data Manager 4.5 database.

Chapter 12 Discusses the procedure for upgrading OFSA 3.5/4.0 version databases to the Financial Data Manager (FDM) database version 4.5.

Chapter 13 Provides information on installing and configuring Oracle Discoverer for use with Financial Data Manager.

Chapter 14 Provides information on the database security framework of the 4.5 Financial Data Manager database.

Chapter 15 Describes the implementation of multi-language support for the FDM database.

Chapter 16 Provides information on how to manage objects within the FDM database environment.

Chapter 17 Provides information on how to manage Leaf Columns within the FDM database.

Chapter 18 Provides information on the duties of the DBA to sustain performance of the FDM database environment.

Chapter 19 Provides information on configuring the Oracle Financial Services Application (OFSA) server-centric software for multiprocessing.

Chapter 20 Provides information on configuring the OFSA Request Queue.

Chapter 21 Describes the FDM Utility scripts and how to use them.

Chapter 22 Provides an instructions for sending copies of your FDM database to Oracle Support Services for assistance.

Appendix A Provides a table that lists the Functional Security values acceptable for the FDM database creation and database upgrade processes.

This reference guide also contains an index.

Related Documents

The following documents provide supplementary information for the subjects discussed in the Installation and Configuration Guide:

- *Getting to Know Oracle8i*
- *Oracle8i Administrator's Guide*
- *Oracle8i Application Developers Guide*
- *Oracle8i Concepts*
- *Oracle8i Designing and Tuning for Performance*
- *Oracle8i Error Messages*
- *Oracle8i Reference*
- *Oracle8i SQL Reference*
- *Oracle8i Utilities*
- *Oracle8i Application Developers Guide*
- *Oracle Discoverer 3.1 Administration Guide*
- *Oracle Discoverer 3.1 User Guide*
- *Oracle Financial Data Manager Administration Guide*
- *Oracle Server SQL Reference Guide*

Conventions

This reference guide uses the following conventions:

Convention	Meaning
. . .	Vertical ellipsis points in an example mean that information not directly related to the example has been omitted.
. . .	Horizontal ellipsis points in statements or commands mean that parts of the statement or command not directly related to the example have been omitted.
boldface text	Boldface type in text indicates a term defined in the text, the glossary, or in both locations.
`bold monospace`	Bold monospace type in text indicates information that you type in.
Italics	Italics emphasize a word or phrase.
< >	Angle brackets enclose user-supplied names (for example, <Branch Name>).
[ ]	Brackets enclose function and terminal keys. In common syntax, brackets denote one or more optional items.
{ }	Braces are used to denote variables, and in command syntax, a choice within a mandatory item. Example of command syntax: *Warning: INIT file {filename} already exists.* Example of choices: {EXIT | QUIT}
->	This arrow indicates a menu path.

Symbols

- Bullets indicate a list of items or topics.

1. Numbered lists are used for sequential steps in completing a procedure.

Orientation of Procedures

Procedures in OFSA reference guides are generally menu-driven rather than command- or icon-driven. Only occasionally is a reference to a toolbar or mouse action necessary because the action has no menu equivalent. If you prefer to use the toolbar icons, refer to Chapter 1, "Introduction".

Notes, Cautions, and Warnings

Certain information may be set off in boxes for purposes of emphasis:

- *Note* refers to interesting but incidental information about the application or information that may be important but of lesser degree than a *Caution* or *Warning*.
- *Caution* indicates the possibility of damage to an application, system, or data.
- *Warning* refers to a situation that is potentially hazardous to people.

Customer Support Information

Customer support is available through Oracle Support Services. Contact your project manager for information about using the support options offered in your geographic region. These options may include the following:

- MetaLink (which provides online access to information about Technical Libraries, Patches, TARs, and Bugs and is available at metalink.oracle.com)
- Telephone support

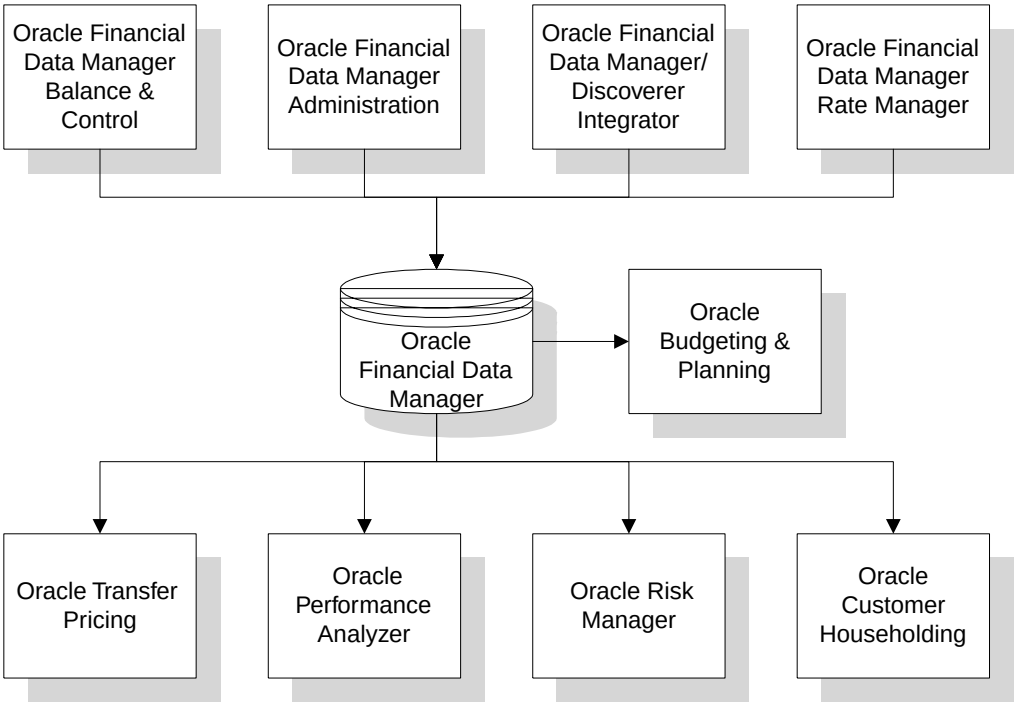
1

Introduction

The Oracle Financial Services Installation and Configuration Guide provides information about installing and configuring the applications within the Oracle Financial Services Applications (OFSA) group of applications. This chapter briefly describes the individual applications within the group.

Oracle Financial Services Overview

The Oracle Financial Data Manager (FDM) is the foundation of the Oracle Financial Services (OFS) group of applications.



OFS Applications

OFS applications form a comprehensive decision support solution that significantly enhances transfer pricing, budgeting and planning, risk management, and performance measurement functions across a financial institution.

Oracle Financial Data Manager

Oracle Financial Data Manager (FDM) is a standalone data warehouse with prepackaged data elements for the financial services industry. FDM is also the foundation for the OFS applications. It provides the database structures necessary to support the individual business applications.

FDM includes Oracle Financial Data Manager Balance & Control, Oracle Financial Data Manager Administration, Oracle Financial Data Manager / Discoverer Integrator, and Oracle Financial Data Manager Rate Manager.

Oracle Financial Data Manager Balance & Control Balance & Control validates, corrects, and aggregates data from the FDM.

Oracle Financial Data Manager Administration FDM Administration manages the FDM, providing security and maintenance capabilities.

Oracle Financial Data Manager/Discoverer Integrator Discoverer Integrator integrates the FDM database with Oracle Discoverer, which provides ad hoc reporting, analysis, and Web publishing capabilities.

Oracle Financial Data Manager Rate Manager FDM Rate Manager manages interest rate, exchange rate, and currency information for the FDM.

Oracle Budgeting & Planning

Budgeting & Planning provides performance-based planning. It integrates cash flow balance sheet and net income forecasting capabilities with the scalability and customizable framework of Oracle Financial Analyzer, part of the Oracle Express group of data access and analysis tools.

Oracle Transfer Pricing

Transfer Pricing calculates a transfer rate for each account and a charge or credit for funds for each asset or liability.

Oracle Performance Analyzer

Performance Analyzer provides comprehensive and flexible cost and equity allocations. It measures product, business unit, and customer profitability.

Oracle Risk Manager

Risk Manager forecasts cash flows, interest income, and market value in order to manage rate risk.

Oracle Customer Householding

Customer Householding provides a fully scalable parallel-processing engine for customer data loading and cleansing, customer relationship linking, customerization, householding, and data aggregation within FDM.

2

New Features and Terminology

This chapter provides an overview of new features and functionality incorporated in the installation and upgrade procedures for the Oracle Financial Services Applications (OFSA) software as well as maintenance and administration. This chapter also highlights content and organizational changes for the *Oracle Financial Services Installation and Configuration Guide*.

New Terminology

The nomenclature used to describe the components and applications that compose the Oracle Financial Services Applications group of applications is changed in Release 4.5.

OFSA versus FDM

Previously, the term *Oracle Financial Services Applications (OFSA)* was used as a generic reference for anything pertaining to these Oracle Financial Services Applications. This is no longer a convention for 4.5. *Oracle Financial Data Manager (FDM)* is now a more specific term for referring to the database environment supporting the OFS applications. The term *OFSA* is used now to refer to the entire group of business applications for Oracle Financial Services.

All of the documentation, including this guide, uses the following naming conventions:

Financial Data Manager (FDM)

Financial Data Manager refers to the database environment and tools that support the Oracle Financial Services Applications group of applications. It consists of the following individual components:

- Financial Data Model

- Processing Data Mart
- Reporting Data Mart
- Oracle Discoverer Integration
- Oracle Reports Starter Kit
- Oracle Balance & Control
- Oracle Financial Data Manager Administration
- Oracle Rate Manager

The term *Financial Data Manager* identifies this collective grouping of components.

The following Oracle Financial Services applications are based upon the FDM environment:

- Oracle Performance Analyzer
- Oracle Risk Manager
- Oracle Transfer Pricing

The following Oracle Financial Services applications integrate with FDM but are not based upon the FDM environment:

- Oracle Budgeting & Planning
- Oracle Market Manager

Oracle Financial Services Applications (OFSA)

This term identifies the entire Oracle Financial Services Applications group of products. This term is used to identify operations that are common to all of the applications in the group.

New Features

This section describes new features relating to software installation, upgrades, and maintenance.

Supported Operating Systems

The client-side of Release 4.5 is configured to run on either Windows NT 4.0 or Windows 95/98.

Financial Data Manager Administration

The Financial Data Manager Administration application is a new application for administering and managing the FDM database environment. This application provides enhanced security and object management functionality, including:

Universal Login

The concept of *Universal Database Login* indicates the ability to log in to the FDM database from any application or SQL compatible tool using a single login account. Previous to Release 4.5, the OFS applications employed password encryption for database security purposes, thereby prohibiting users from logging into reporting tools such as Oracle Discoverer with the same login account used for logging into the OFS applications. FDM 4.5 does not employ user password encryption, enabling administrators to create a single login account for each user.

Enhanced Security Features

FDM Administration provides enhanced security management capability for managing the FDM environment. This includes the following:

- Database Security using Roles
- Division of Security Management Responsibilities
- User Groups and Security Profiles
- Dynamic Privileges

For more information about the security management functionality provided by the FDM Administration application, refer to the *Oracle Financial Data Manager Administration Guide.*

Enhanced Object Management Features

FDM Administration provides new features for customizing and extending your FDM database. This includes the following

- Direct Integration with Oracle RDBMS Catalog
- Object Registration
- Table Classifications
- Description Table Mapping

For more information about the security management functionality provided by the FDM Administration application, refer to the *Oracle Financial Data Manager Administration Guide.*

Database Management Utilities

FDM provides utilities to facility managing the FDM database environment. These utilities include:

- Template scripts for Financial Instrument table creation
- Procedure for altering Balance column definitions
- Procedure for mapping old 3.5/4.0 Currency Codes to the new ISO Currency Code implementation.
- Procedure for synchronizing Leaf values and Code Descriptions

Rate Manager

Rate Manager manages interest rate and exchange rate information for the Financial Data Manager database, improving upon and replacing Historical Rates ID. Rate Manager is a component of Financial Data Manager.

Multi-Currency Support

The FDM database environment supports the use of multiple currencies. Financial values such as account balances and transaction fees can be stored within the database in multiple currencies for use with OFS application reporting and processing operations.

Multi-Language Support (MLS)

The FDM database environment supports the use of multiple languages (otherwise known as Multi-Language Support, or MLS). This feature allows multiple users of different languages to retrieve information in their own language from the same FDM database.

Functionality Changes in Risk Manager and ID Conversions

Functionality changes in Risk Manager require the conversion of data for the following three IDs: Prepayment, Forecast Balance, and Process. The ID conversion routines occur as part of the database upgrade process.

Document Changes

The *Oracle Financial Services Installation and Configuration Guide* includes technical information on processes pertaining to maintenance and administration of the FDM and Budgeting & Planning database environments. It also includes information on installing, upgrading and configuring these database environments as well as the

Oracle Financial Services applications. New sections for Release 4.5 include the following:

Installing and Configuring Discoverer

Procedures for installing the OFSA Discoverer Standard Reports Workbooks are included in Chapter 13, "Installing and Configuring Discoverer". Previously, the information in this chapter was presented in both the Database Installation and Database Upgrade Process chapters.

Upgrading from OFSA 3.5/4.0

This chapter provides information essential for users upgrading from a previous version of the OFSA group of applications. It explains in detail how the upgrade process migrates objects and security privileges to the new 4.5 FDM database environment. Review this chapter prior to beginning your upgrade procedure.

Database Security

This is a new chapter that includes information regarding how FDM implements database security. The security implementation in FDM 4.5 is significantly different from that of OFSA 3.5/4.0.

Multi-Language Support

This is a new chapter detailing how Multi-Language Support is implemented within FDM 4.5.

Object Management

This is a new chapter detailing how to manage the different types of database objects within the FDM database. Previously, the information in this chapter was included in the chapter titled Database Administration.

Leaf Management

This is a new chapter detailing how to add and remove Leaf Columns from your FDM database.

Database Tuning

This chapter provides information on how to tune your FDM database for optimal performance. Previously, the information in this chapter was included in the chapter titled Database Administration.

Sending Databases to Oracle Support Services

In some circumstances, Oracle Support Services may request a copy of your database in order to replicate a problem or perform some troubleshooting. This chapter provides instructions on how to send databases to Oracle Support Services for these purposes.

3

Certifications

This chapter provides details on the platforms, operating systems, and software for which Release 4.5 of the Oracle Financial Services Applications (OFSA) group of applications has been certified. The specific topics addressed in this chapter include:

- Server-Side Certification Statement
- Client-Side Certification Statement

Caution: The use of non-certified components can cause unexplained or undesirable system behavior, such as General Protection Faults or core dumps. Contact the your hardware or software vendor regarding questions of compatibility and performance.

Server-Side Certification Statement

The following table identifies the server-side, qualified hardware and software components for this release.

Server-Side Certifications for All OFSA Applications Except Budgeting & Planning

Server	Operating System	Database
Sun	Solaris 2.6* with patch 105591-02 and Solaris 2.7*	Oracle8i R2 (8.1.6.x)
HP (9000 Series)	HP-UX 11.0*	Oracle8i R2 (8.1.6.x)
IBM R/S 6000	AIX 4.3.3*	Oracle8i R2 (8.1.6.x)
Compaq	Tru64 UNIX 4.0E*	Oracle8i R2 (8.1.6.x)

*All operating system certifications include all relevant Y2K patches from the vendor.

Server-Side Certifications for Budgeting & Planning

Server	Operating System	Database	Listener
NT	4.0	Express Server 6.3.0.1	Oracle Application Server 4.0.8.1
HP (9000 Series)	HP-UX 11.0	Express Server 6.3.0.1	Oracle Application Server 4.0.8.1
IBM R/S 6000	AIX 4.3.3	Express Server 6.3.0.1	Oracle Application Server 4.0.8.1
Compaq	Tru64 UNIX 4.0E	Express Server 6.3.0.1	Oracle Application Server 4.0.8.1

Client-Side Certification Statement

The following table identifies the client-side, certified software components for this release.

OFSA Applications/Functionality	Operating System/ODBC/Database Drivers
All applications/functionality	■ Win NT 4.0 and minimum Service Pack 3 or Windows 95/98 SR2*
All applications (except FDM Administration and Discoverer Integrator)	■ 16-bit SQL*Net Client version 2.3.3.0.1
Budgeting & Planning	■ Financial Analyzer 6.3.0.0 MLE w/Patch #6325_2* ■ Oracle Express Web Agent 6.3.0.1 MLE w/Patch #owa630_p1* ■ Oracle JInitiator 1.1.7.29 or higher*
Discoverer Integrator	■ Oracle NET8 8.0.4.0.2c
FDM Administration	■ 32-bit SQL*Net Client version 2.3.4.0.2 ■ Oracle Developer Forms runtime 6.0
Import/Export functionality for: ■ Balance & Control ■ Performance Analyzer ■ Portfolio Analyzer ■ Risk Manager ■ Transfer Pricing	■ 16-bit Merant version 2.5.3 GA-dbase IV*
Knowledge engines for the following applications: ■ Balance & Control	■ 16-bit SQL*Net Client version 2.3.3.0.1
Knowledge engines for the following applications: ■ Performance Analyzer ■ Risk Manager ■ Transfer Pricing	■ Oracle NET8 8.0.4.0.2c

OFSA Applications/Functionality	Operating System/ODBC/Database Drivers
Oracle Discoverer Standard Reports	■ Oracle Discoverer 3.1.36*
Oracle Reports Standard Reports	■ Oracle Reports 6.0*

*Non-Oracle and Oracle Products (such as Oracle Discoverer) not included on the Oracle Installer are categorized as *Third-Party Components*. Such components are marked with an asterisk (*) in the table. You need to install these products separately because they are not included on the OFSA CD.

4

System Description

The Oracle Financial Services Applications (OFSA) group of applications is designed to run in a three-tier client/server environment. This chapter provides an overview of the application's three primary components and the connectivity between them.

The following topics are included in this chapter:

- Application Components
- System Environment
- Database Connectivity
- Database Description

Application Components

The OFSA applications are designed around the following three interrelated components.

Database Component

The database for this application is an instance with the database schema and system tables pre-loaded according to the logical data model. This is referred to as the Oracle Financial Data Manager (FDM) database.

Server-side Component

The server-centric component is a set of application modules (shared libraries and executables) that run on a UNIX server.

Client-side Component

The client-side component (for either a single or multiple workstations) is a set of application modules (.DLLs and .EXEs) running on a Windows-based operating system. These modules communicate with the server-side of the application and the FDM database through SQL*Net and/or NET8.

System Environment

The OFSA group of applications operates within a three-tier client/server environment and is capable of providing enterprise-wide operation through a LAN and WAN.

Three-tier Client/Server Environment

The OFSA engines and FDM database can both be placed on the same server or on separate servers. This is an optional consideration, dependent upon your organization's needs. Some organizations choose to have one server dedicated to the OFSA engines and another dedicated to the FDM database, while others use one server for both.

Data Sourcing Environment

Data sourcing, as well as sharing, between multiple clients occurs through the network connection to the server or servers dedicated to the server-side application and database. Relevant data can be extracted from multiple sources and loaded into the Oracle database. This data is used by the application-specific processes, such as running calculation engines or generating reports.

Database Connectivity

Connectivity to the Oracle database requires two communication applications.

A Network Protocol

A network protocol supplies the communication link between computer systems. Protocol services handle most network events, errors and security and operate on dissimilar types of computer systems.

SQL*Net and NET8

SQL*Net and NET8 are the communication components used by Oracle to share information stored in different Oracle databases. Oracle tools and third party

applications running on one system can manipulate databases located on other systems on a network.

Both components provide interfaces for applications to connect to databases over many different network protocols; TCP/IP is one of the protocols supported.

SQL*Net and NET8 use server processes to listen for client connection requests and then connect the client to the correct database. SQL*Net version 2 and NET8 are the current listeners and use Transparent Network Substrate (TNS) to provide client/server connectivity. This design provides applications with a single, common interface to all industry-standard protocols.

Database Description

The FDM database contains data extracted from your organization's mainframe that the knowledge engines use when performing their functions.

5

System Requirements

This chapter describes the recommended hardware requirements for the client side of the three-tier client/server environment.

Client-side Requirements

The recommended common client components include:

1. IBM compatible Pentium 200 Mhz PC
2. SVGA Monitor
3. 32 MB RAM (64 MB RAM is recommended if you are using the Monte Carlo functionality in Risk Manager)
4. 300 MB free disk space
5. Access to a CD-ROM drive

6

UNIX Server Installation and Configuration

This chapter provides information on installing the Oracle Financial Services Applications (OFSA) server-centric application and properly configuring your UNIX server.

The following topics are covered in this chapter:

- Preparing Your Server for Installation
- Installing the OFSA Server-Centric Application
- Configuring OFSA Server-Centric Applications
- Determining Shared Resource Requirements
- Adjusting UNIX Kernel Parameters
- Determining Application-Specific Memory Requirements
- Configuring the Request Queue Log File
- Other Configuration Issues

Preparing Your Server for Installation

When configuring Oracle applications for a specific hardware platform, you need to set parameters specific to that platform. Refer to the Oracle installation documentation for your server for additional information regarding these parameters.

Installation Choices

The OFSA server instance creation provides a great deal of flexibility. While Oracle does not require a specific method for creating an Oracle instance, it is

recommended that you follow the guidelines for installing and creating an OFSA Oracle instance as described in the Oracle Optimal Flexible Architecture standard document. The standard is designed to enhance ease of maintenance. It also reduces potential ambiguities by utilizing structured naming conventions.

There are many choices to be made, even within the standard, such as (but not limited to):

- Raw Devices versus File Systems
- Striped Disk Devices versus Concatenated Disk Devices
- ASYNC I/O versus Multiple DB Writers

It is recommended that you carefully consider the consequences of selecting certain options before deciding on how you want to create an instance on the OFSA server. Generally the options you need to balance are speed of operation versus ease of use and maintenance. Read Chapter 16, "FDM Object Management" and Chapter 18, "FDM Database Performance Management" before you make a decision will help you to understand these considerations.

Prior to Installation

Before you install the server-centric applications, review the storage requirements for each application and the permissions that will be necessary.

Storage Requirements

The following table lists the OFSA storage requirements by platform:

Category	Sun	HP	AIX	Compaq
Applications	80 MB	85 MB	50 MB	85 MB
Database Scripts	20 MB	20 MB	20 MB	20 MB
Workspace[1]	25 MB	25 MB	25 MB	25 MB
Total	125 MB	130 MB	100 MB	130 MB

[1] Default value for workspace based on one instance of Request Queue and one database instance

Workspace storage is calculated based upon one active Request Queue (there is always one Request Queue per database instance). The workspace calculation

further assumes that the Request Queue's maximum logfile control parameters are set to default (MaxFileSize = 512kb, IdealFileSize = 256kb). See Chapter 20, "Request Queue" for more information about these default settings.

The suggested formula for calculating the required workspace is as follows:

```
workspace = (number of Request Queues) * (20 Mb + (2 * max. logfile size) )
```

Required User and Group

Before installing the applications and if they do not already exist, create the user ofsa and the group ofsadba.

Installing the OFSA Server-Centric Application

This section provides instructions on installing the OFSA server-centric applications. The servers included in this section have been certified for this release. See Chapter 3, "Certifications" for additional information on certified servers.

Installation routines are provided for the following servers:

- Sun
- Hewlett Packard
- IBM-AIX
- Compaq

> **Note:** In this chapter, OFSA_INSTALL is the convention used to indicate where OFSA is installed in your directory structure. Also, all references to owner are to the UNIX account that will own all of the software files.

Installing OFSA on Sun Servers

Sun servers use the pkgadd command to install software packages. Complete the following steps to install the OFSA server-centric applications on these servers.

1. Log in as the root user.
2. Insert the CD-ROM into the CD-ROM drive.
3. Mount the CD-ROM to a UNIX mount point. An example of this command follows:

```
mount /dev/dsk/c2t2d0 /cdrom
```

4. If you want to keep a previously installed OFSA version, you may need to create an alternate admin file. This will be required if a prompt appears asking which version of OFSA to overwrite.

Caution: Pkgadd will not operate properly if you do not create an alternate admin file when this prompt appears.

See the section entitled "Installing Multiple Versions of OFSA on Sun" for instructions on keeping previous OFSA versions.

---

5. If you want to remove a previously installed OFSA version, refer to the platform-specific system administration documentation for details on how to remove a package.

Caution: When removing packages (`pkgrm`), be careful not to remove packages in addition to the OFSA server-centric applications. This can occur when you invoke a package remove statement. Refer to your UNIX documentation so that you remove only the OFSA group of applications.

---

6. In UNIX, change the directory to where the CD-ROM is mounted. An example of this command follows:

```
cd /cdrom
```

7. In UNIX, change the directory to the appropriate platform by typing the following command.

```
cd server/<server name>
```

8. Install the package by typing the following command. Note that the package name is case sensitive.

```
pkgadd -d ‘pwd‘ ORCLofsa
```

Note: If you do not type a package name and click Enter, a list appears with package choices. Select the appropriate package name and click Enter.

The package begins installing the server-centric applications, displaying these prompts:

a. The following prompt appears:

```
Install ofsa where? (default: /db/d00/ofsa)
```

Click Enter to accept the default directory or type a new installation directory and click Enter.

b. The following prompt appears:

```
Install applications (a), scripts (s), BP web client (b), or full
package (default: f)?

specify one of the options (default: f)? [ascbf]
```

Select a full installation (f) by clicking Enter.

c. The following prompt appears:

```
Before installing OFSA Web Applications, you need to have installed
a web server on this machine.

Do you have a web server installed on this machine?  [y,n,?,q]
```

The Budgeting & Planning web client (Oracle Budgeting & Planning) requires a web server on the machine. If you select *No* at this prompt, the installation continues and displays the following message:

```
Removing bpweb from the install and continuing if other categories were
selected.
```

d. If you designate *Yes* for having a web server installed on the machine, the following prompt appears:

```
Where should web application files be installed?

(specify a directory in the web servers document tree where the BPweb
files may be copied.  This will create a directory there called bpweb)

Write permission to that directory for  is required (default:
/usr/etc/httpd/docs/ofsa)
```

Designate the appropriate directory and click Enter to continue.

e. The following prompt appears:

```
Owner for ofsa files? (default: ofsa)
```

All files and directories created by the pkgadd command are assigned to the owner specified in this step.

Click Enter to accept the default owner or type a new owner name and click Enter.

f. The following prompt appears:

```
What group should the installed files belong to? (default: ofsadba)
```

All files and directories created by the pkgadd command are assigned to the group specified in this step.

Click Enter to accept the default group or type a new group name and click Enter.

9. After the installation is complete, a message similar to the following message appears:

```
Installation of <ORCLofsa> was successful
```

This message indicates that you have successfully installed the OFSA server-centric applications on your server and that the installation process has created the following directories in the installation directory:

Directory	OFSA Application Description
../etc	.INI and other miscellaneous files
../bin	Executables
../lib	Shared libraries
../log	Empty directory (Request Queue workspace)
../dbs	Database change and create scripts
../mm	Market Manager

Installation of Budgeting & Planning Web Server Components

There are no additional steps required for installing the Budgeting & Planning Web Server Components on a Sun server. The pkgadd command automatically installs

the required components when you install the Budgeting & Planning Web Server in step 8.

> **Note:** If you do not install Budgeting & Planning Web Server on initial install of OFSA but afterwards want to install it on your Sun server, follow the instructions described in the Installation of Budgeting & Planning Web Server Components sub-section of the Installing OFSA on a Hewlett Packard Server. The instructions for adding Budgeting & Planning Web Server to an existing OFSA installation for Sun are the same as installing Budgeting & Planning Web Server on a Hewlett Packard server.

Installing Multiple Versions of OFSA on Sun

Multiple versions of the OFSA applications can be installed on the same server. However, each version needs to be installed in a different directory. Create these directories when you are installing the current OFSA package.

This is done by creating an admin file and including the `-a` option in pkgadd. The steps for creating the admin file follow.

> **Caution:** You cannot install the same version of the OFSA server-centric application on your server more than one time.

Although you cannot install the same version of the application on a server more than once, you can simultaneously install and run the same general release version of the application if each version has a different patch. For example, you can run Release 4.5 maintenance patch 02 and Release 4.5 maintenance patch 03 on the same server.

To create an admin file that enables you to install multiple versions of OFSA complete the following steps.

1. Log in as the root user.
2. In UNIX, change the directory to /var/sadm/install/admin by typing the following command:

   ```
   cd /var/sadm/install/admin
   ```

3. Type the following command.

   ```
   sed s/instance=.*/instance=unique/ default > ofsa
   ```

The admin file that enables you to install multiple versions of OFSA is now created.

4. Type the `-a` option to pkgadd to include the admin file in the installation command. An example of this command follows:

```
pkgadd -d 'pwd' -a ofsa
```

Note: Once you create the admin file, you do not need to recreate it every time you install this release of the OFSA server-centric applications.

Installing OFSA on a Hewlett Packard Server

Hewlett Packard (HP) servers use the swinstall program to install software packages. The swinstall program runs on several terminal types including character based (vt type) and X Windows (GUI). Both provide a command-line based mechanism to install the software.

The command-line based installation mechanism is documented in this section. See the appropriate HP system administration documentation if you plan to use the character based mechanism or GUI.

Complete the following steps to install the OFSA server-centric applications on your HP server.

1. Log in as the root user.
2. Create the following directory:

   /cdrom

3. Insert the CD-ROM into the CD-ROM drive.
4. Check the file **/etc/pfs_fstab** for the following entry (replacing the device name where appropriate):

   /dev/dsk/c3t2d0 /cdrom pfs-rrip xlat=unix 0 0

5. Install and run the following daemons by typing these commands:

```
nohup /usr/sbin/pfs_mountd &
nohup /usr/sbin/pfsd &
```

Note: Once these daemons have been installed you do not need to run them again to mount a CD-ROM.

6. Mount the CD-ROM by typing the following command:

   ```
   /usr/sbin/pfs_mount /cdrom
   ```

7. Define a UNIX account and group for the OFSA applications. HP requires that both the account and group be defined before running the installation process. The following options are available in defining your UNIX account and group.

   - If you have installed a previous version of OFSA applications and you want to keep this version on your server then the account and group set up in the default directory (`default /db/d00/ofsa`)is used for the installation.
   - If you are not keeping a previous version of the application or you are installing the OFSA application for the first time, you have the option of either using the default UNIX account, which is ofsa, and the default UNIX group, which is ofsadba, or creating a UNIX account and group that follows your organization's naming conventions.

     If you create your own UNIX account and group then you also need to either create the following directory, `default /db/d00/ofsa,` or create your own directory and pass that directory name to the swinstall command. The installation process looks to this directory for the account and group that will be used for the OFSA group of applications.

Caution: If you do not create the UNIX account and UNIX group before installation, your installation will fail.

8. If you want to keep a previously installed OFSA version you must specify the following switch to the swinstall command:

   ```
   -x allow_multiple_versions=true
   ```

9. If you want to remove a previously installed OFSA version refer to the HP system administration documentation for further information on removing a package.

10. In UNIX, change the directory to where the CD-ROM is mounted. An example of this command follows:

```
cd /cdrom
```

11. In UNIX, change the directory to the appropriate platform by typing the following command:

```
cd server/HP
```

12. If you are installing the package and not retaining previous versions or do not have previous versions installed type the following command:

```
swinstall -s ‘pwd‘ -x mount_all_filesystems=false
OFSA:/db/d00/app/ofsa/ofsa4.5-092-hp
```

The following portion of this command, `:/db/d00/app/ofsa/ofsa4.5-092-hp`, is the installation (OFSA_INSTALL) directory location. If you have chosen a different location, replace the default location at this point in the command line with your directory location.

The package begins installing the server-centric application.

13. If you are installing the package and keeping a previous version, type the following command:

```
swinstall -s ‘pwd‘ -x allow_multiple_versions=true
-x mount_all_filesystems=false OFSA:/db/d00/app/ofsa/ofsa4.05092-hp
```

The following portion of this command, `:/db/d00/app/ofsa/ofsa4.5-092-hp`, is the installation (OFSA_INSTALL) directory location. If you have chosen a different location, replace the default location at this point in the command line with your directory location.

The package begins installing the server-centric application.

14. After the installation is complete, a message similar to the following appears:

```
Installation of <ORCLofsa> was successful
```

This message indicates that you have successfully installed the OFSA server-centric application on your server and that the installation process has created the following directories in the installation directory:

Directory	OFSA Application Description
../etc	.INI and other miscellaneous files

Directory	OFSA Application Description
../bin	Executables
../lib	Shared libraries
../log	Empty directory (Request Queue workspace)
../dbs	Database change and create scripts
../mm	Market Manager

When finished, unmount the CD-ROM on the HP UX 11.0 server by typing the following command:

```
/user/sbin/pfs_umount /cdrom
```

Installation of Budgeting & Planning Web Server Components

If you are using Budgeting & Planning, then you need to install the Budgeting & Planning Web Server components. To do this, complete the following steps:

1. Run the bpweb_setup.sh script:

   The default location for the bpweb_setup.sh script is the OFSA_INSTALL /dbs/<OFSA release>/bpweb subdirectory of your OFSA installation directory. In this chapter, OFSA_INSTALL is the convention used to indicate where the OFSA applications are installed in your directory structure.

   To run the script, type the following:

   ```
   ./bpweb_setup.sh
   ```

2. The script provides the following prompt:

   ```
   Before installing OFSA Web Applications, you need to have a web server
   installed on this machine.

   Do you have a web server installed on this machine? (yes or no)
   ```

   The Budgeting & Planning Web Server components require that a web server is installed on the machine. If you do not have a web server installed, type `no` and click Enter and the installation will terminate. If you do have a web server installed, type `yes` and click Enter to continue with the installation.

3. The script then prompts for the location to install the Budgeting & Planning Web Server components.

   ```
   Where should web application files be installed?
   ```

(Specify a directory in the web servers document tree where the BPweb files can be copied.

Write permission to that directory for ofsa is required (default: /usr/etc/httpd/docs/ofsa)

Specify the appropriate location. You must have write permission to the directory specified.

Installing OFSA on an IBM-AIX and Compaq Alpha Server

To install the OFSA server-centric applications on your IBM-AIX or Compaq Alpha server complete the following steps.

1. Log in as the root user.
2. Insert the CD-ROM into the CD-ROM drive.
3. Mount the CD-ROM to a UNIX mount point. An example of this command follows:

   ```
   mount -v cdrfs -r /dev/cd0 /cdrom
   ```

4. Log in as the UNIX account that will own the OFSA server-centric applications. The default account is ofsa
5. Create the directory to contain the OFSA server-centric applications. An example of this command follows:

   ```
   mkdir /db/d00/app/ofsa/ofsa4.5-092-aix
   ```

6. In UNIX, change the directory to where the CD-ROM is mounted by typing the following command:

   ```
   cd /cdrom
   ```

7. In UNIX, change the directory to the appropriate platform by typing the following command:

   ```
   cd server/IBM
   ```

8. Copy the software package containing the OFSA server-centric applications to your designated installation directory. An example of this command follows:

   ```
   cp ofsa4.5-092-aix.tar /db/d00/app/ofsa/ofsa4.5-092-aix/
   ```

   This process requires 100 MB of temporary space. You can reclaim this space after you complete the installation process by removing the tar file.

9. In UNIX, change the directory to the OFSA_INSTALL directory. An example of this command follows:

   ```
   cd /db/d00/app/ofsa/ofsa4.5-092-aix
   ```

10. Extract the package by typing the following tar command with the **x** (extract) **v** (verbose) **f** (file) options:

    ```
    tar -xvf ofsa4.5-092-aix.tar
    ```

11. To complete the installation, run the postinstall script from the OFSA_INSTALL directory. For example:

    ```
    cd /db/d00/app/ofsa/ofsa4.5-092-aix

    ./postinstall
    ```

Installation of Budgeting & Planning Web Server Components

If you are using Budgeting & Planning, then you need to install the Budgeting & Planning Web Server components. To do this, complete the following steps:

1. Run the bpweb_setup.sh script:

   The default location for the bpweb_setup.sh script is the OFSA_INSTALL /dbs/<OFSA release>/bpweb subdirectory of your OFSA installation directory. In this chapter, OFSA_INSTALL is the convention used to indicate where the OFSA applications are installed in your directory structure.

   To run the script, type the following:

   ```
   ./bpweb_setup.sh
   ```

2. The script provides the following prompt:

   ```
   Before installing OFSA Web Applications, you need to have a web server
   installed on this machine.

   Do you have a web server installed on this machine? (yes or no)
   ```

   The Budgeting & Planning Web Server components require that a web server is installed on the machine. If you do not have a web server installed, type `no` and click Enter and the installation will terminate. If you do have a web server installed, type `yes` and click Enter to continue with the installation.

3. The script then prompts for the location to install the Budgeting & Planning Web Server components.

   ```
   Where should web application files be installed?

   (specify a directory in the web servers document tree where the BPweb files
   ```

```
can be copied.)

Write permission to that directory for ofsa is required (default:
/usr/etc/httpd/docs/ofsa)
```

Specify the appropriate location. You must have write permission to the directory specified.

Creating and Locating the OFS.INI File

The installation program creates the **ofs.ini** file and locates it in the <OFSA_INSTALL>/etc directory. This file provides a list of datasources available within the OFSA group of applications and specific data unique to each datasource.

Components of the OFS.INI File

The following table provides an example of an ofs.ini file:

```
[Oracle_Example]
DriverType              = ORACLE
ServerName              = DB name
Database                = $ORACLE_SID
```

The following table describes the components of the ofs.ini file.

OFS.INI Component	Description
[Oracle_Example]	This heading indicates that the information entered pertains to this datasource only. This is the name you provide to the server-based applications when prompted for a database.
DriverType	This indicates the database driver to use to connect to the datasource. This should always be “ORACLE” and is not case sensitive.
ServerName	Type the database alias here. This is the database name that SQL*Net or NET8 recognize. Refer to Chapter 7, "Client Software Installation and Configuration" for additional information on this topic. If this is blank, the $ORACLE_SID environment variable is used to make a local connection. If the OFSA applications and database are installed on the same server, this is the preferred method.

OFS.INI Component	Description
Database	Normally the same as ServerName. If ServerName is blank, this should be the value of the $ORACLE_SID.

The installation program creates a sample ofs.ini file. You can either modify this sample file or create your own.

Configuring OFSA Server-Centric Applications

This section explains the configuration of OFSA and UNIX kernel parameters. Use this section as a guide to determine OFSA's usage of system resources and to properly configure the kernel and .INI file. It is important to understand the production processes and normal usage of each OFSA application to properly configure the server-centric applications.

The following terms are used in this section.

Term	Definition
Bulk Processing	A method of processing using an update statement to do calculations and update multiple rows.
Common COA leaf – common_coa_id	A column in the OFSA database.
Process	A UNIX process.
Product Leaf	The product leaf as specified in the Configuration ID for Transfer Pricing or Risk Manager. In Balance & Control and Performance Analyzer the product leaf is common_coa_id.
Org Leaf	The organizational leaf as specified in the Configuration ID for Transfer Pricing or Risk Manager. In Balance & Control and Performance Analyzer org_unit_id is the organizational leaf in the OFSA database.
Row by Row Processing	A method of processing using a select statement to fetch data, calculate results using that data and then either write a single row back to the database or aggregate it into multiple rows.
Semaphore	A UNIX resource that exists separate from any process and provides a mechanism for synchronizing processes and controlling access.
Shared Memory	A UNIX resource that exists separate from any process and provides a mechanism for processes to share data.

Term	Definition
Shared Resource	Generally any resource that is shared among multiple processes. For the purposes of this document, a Shared Resource is specifically a Semaphore or a Shared Memory Segment.
OFSA Job	A run of an OFSA ID. An OFSA job can use one or more processes.
Unit of Work	Each OFSA job divides its required data into units of work which are then distributed to processes. A unit of work is a set of rows from the database.

Application .INI Settings

This section describes settings in the application .INI files that affect processing. In each case, guidelines are presented as how to best determine these settings. In most cases it is better to have the settings discussed in this section set too large rather than too small. It is recommended that you set these numbers higher than their minimum expected value, to accommodate growth.

Once you have installed the OFSA server-centric applications on the appropriate server, the installed applications need to be configured properly. The following sections describe specific settings you need to check. The .INI files referenced in these sections are located in the /.../OFSA_INSTALL/etc directory on the server. There are similar sets of .INI files on each client machine. Use and configuration of the client .INI files is discussed in Chapter 7, "Client Software Installation and Configuration".

Configuring Paths

The following information is required to be in the ofs.ini file. This information assumes that /db/d00/ofsa is the directory where OFSA is installed.

```
[OFSRQ]

BC               = /db/d00/ofsa/bin/ofsbc
RQTEST           = /db/d00/ofsa/bin/ofstest
PRW              = /db/d00/ofsa/bin/ofspa
TPW              = /db/d00/ofsa/bin/ofstp
TMW              = /db/d00/ofsa/bin/ofsrm
OME              = /db/d00/ofsa/bin/ofste
WorkingDirectory = /db/d00/ofsa/log
```

The OME setting is for the Transformation ID. The last setting, WorkingDirectory, specifies the directory that Request Queue sets as its current directory. If the

standard procedure is followed, this is the /db/d00/ofsa/log directory. This setting defines the location of all log files and Request Queue output.

Ledger_Stat Buffer Size

This section applies to all users of Performance Analyzer and users of Transfer Pricing that select the Ledger_Stat option. This setting affects performance. Setting it too small can result in significant degradation in performance while setting it too large uses an excessive amount of system memory.

By default the buffer is set at 2,000 elements.

Calculating the Optimal Setting

The easiest way to determine the optimal setting for these numbers is to first do a full processing run, then execute the application-specific statements. Setting the size larger has no effect on performance, however, the buffer size should be set larger than these values to provide for future growth. Replace <current_year> with the appropriate value.

Performance Analyzer SQL Statements

For Performance Analyzer execute the following statement on Oracle:

```
select max(count(*)) from ledger_stat where identity_code in (select
distinct identity_code from ofsa_catalog_of_ids, ofsa_data_identity where
to_char(ofsa_catalog_of_ids.sys_id_num)=ofsa_data_identity.description and
ofsa_catalog_of_ids.id_type=0) and year_s = <current_year> group by
identity_code;
```

Transfer Pricing SQL Statements

If Ledger_Stat is being used for Transfer Pricing execute the following statement on Oracle:

```
select max(count(*)) from ledger_stat where identity_code in (select
distinct identity_code from ofsa_catalog_of_ids, ofsa_data_identity where
to_char(ofsa_catalog_of_ids.sys_id_num)=ofsa_data_identity.description and
ofsa_catalog_of_ids.id_type=204) and year_s = <current_year> group by
identity_code;
```

Changing the .INI Setting

The .INI setting for Performance Analyzer is located in the **ofspa.ini** file. For Transfer Pricing, it is in the **ofstp.ini** file. The entry is as follows (shown with default setting):

```
[Ledger_Stat]
BufSize = 2000
```

Transfer Pricing Migration Buffer Size

This section applies only if your organization is using Transfer Pricing for transfer pricing Ledger_Stat and migration of detail rates to Ledger_Stat. Establishing this setting larger than necessary will have no affect on performance. Establishing it smaller than necessary will result in an error message from Transfer Pricing that it has run out of space.

Determining the Necessary Space for Migration

Perform the following steps calculate the amount of space needed for migration.

1. Determine the detail tables that will have transfer pricing results migrated to Ledger_Stat.
2. Determine the number of unique org / common / application combinations in each table. In some instances, the common and application dimensions can be the same.
3. Determine the number of org / common / application combinations in Ledger_ Stat for financial element 140 that do not exist in any detail tables.
4. The sum of the values from steps 2 and 3 is the total migration buffer size required. SegCount * SegSize should be greater than this number to provide for future growth.

SegCount and SegSize

The migration buffer operates by allocating a shared memory segment of size 108 * SegSize bytes. When the segment fills with data, it will allocate an additional segment of that size until SegCount segments have been allocated. If the application attempts to place additional data into the migration buffer, an error condition occurs and no more data is placed in the buffer.

Determining how large to make SegCount and how small to make SegSize depends on how you are using Transfer Pricing. If Transfer Pricing uses a single process to do all migration, set SegCount to 1 and SegSize to the number calculated in step 4. In the event that smaller migration runs are also performed, it may be desirable to set SegSize to some fraction of the size and set SegCount to an appropriate value. SegCount can be no greater than three less than the maximum number of shared memory segments that can be attached to a process. In general, SegCount should probably never be greater than 10.

Changing the .INI Setting

The .INI setting for Transfer Pricing is located in the ofstp.ini file. The entry is as follows (shown with defaults):

```
[Migration]
SegSize = 10000
SegCount = 5
```

Upsert Method

The Upsert Method parameter controls how updates and inserts are applied to the LEDGER_STAT table for Performance Analyzer allocation processing.

Previously, Performance Analyzer executed all allocation updates and inserts directly on the LEDGER_STAT table as the input data was processed. OFSA 4.5 now automatically uses temporary tables to store output from the process if an allocation satisfies all of the following conditions:

- The Allocation ID filters on the LEDGER_STAT table OR calculates Percentage Distribution based on the LEDGER_STAT table.
- The Allocation ID Debits or Credits to the LEDGER_STAT table.

You can use the temporary tables to store output for other types of allocations that output to the LEDGER_STAT table by specifying the Upsert Method globally in the ofspa.ini file on the server.

Specify the Upsert Method in the ofspa.ini file as follows:

```
[LEDGER_STAT]
UpsertMethod=2;
```

Where 0 means never use temporary tables for allocations other than the kind that satisfies the conditions already described, 1 means always use temporary tables and 2 (or greater) means the same as 0.

Shared Memory .INI Setting

This section provides the information necessary to calculate the approximate amount of shared memory needed by Risk Manager, based on the nature of the IDs you are running. The default shared memory setting is 16MB, which is the suggested minimum shared memory setting. However, the memory setting for your institution is determined by the formulas in this section and can be greater or lesser than the default setting.

The implementation of Risk Manager allocates memory in blocks that are powers of two and rounds the shared memory setting up to the next power of two.

Calculating the Shared Memory Usage

To calculate the minimum required size of this setting, you need the following information.

Component	Descriptions
NUMBUCK	Number of Static Buckets
NUMEVENTS	Maximum number of events (such as repricing or payments) generated for an instrument record
NUMFINELEM	Number of financial elements generated in a run
NUMGBUCK	Number of Gap Buckets
NUMGSDATES	Number of Gap Start Dates
NUMORG	Number of organizational unit leaves
NUMORGPROD	Number of unique combinations of ORG and PROD in all processed instrument tables
NUMPROCS	The NumProcesses setting from the ofsrm.ini file
NUMPROD	Number of product leaves used in a run
NUMSCEN	Number of Scenarios
NUMSDATES	Number of Start Dates
NUMROLLINTO	Number of RollInto leaves
NUMUNITS	The number of units of work to be processed

Scenario-based Run

The formula for calculating the minimum size of the shared memory segment for a scenario-based run is:

```
(NUMROLLINTO + 4 * NUMPROCS) *
(8 * NUMBUCK * NUMFINELEM * NUMSCEN +
8 * NUMGSDATES * 10 * NUMBUCK * NUMSCEN
)+
16 * NUMORGPROD +
NUMROLLINTO * NUMBUCK * (NUMSCEN * 225 + 32)
```

if autobalancing:

```
+ 25 * 8 * NUMSCEN * NUMBUCK
```

Stochastic-based Run

The formulas for calculating the minimum size of the shared memory segment for the stochastic runs are listed, showing the formula combinations for different run scenarios:

The basic stochastic-run formula is as follows:

```
1200 * 2 * NUMPROCS +
16 * NUMORGPROD +
```

if Market Value only run (formula A):

```
+ 2 * NUMPROCS * 50
```

if Value at Risk (VAR) run (formula B):

```
+ 2 * NUMPROCS * (50 + 8 * (NUMSCEN + 1))
```

if Earnings run:

```
+ NUMROLLINTO * NUMBUCK * (NUMSCEN * 225 + 32) +
NUMBUCK * NUMSCEN * 20 * 8
```

if Portfolio Earnings is on:

```
+ 16 * NUMORGPROD +
+ 16 * NUMBUCK * (1 + NUMSCEN)
```

if Autobalancing is on:

```
+ 25 * 8 * NUMSCEN * NUMBUCK
```

if Earnings is combined with Market Value only:

```
+ Formula A
```

if Earnings is combined with VAR:

```
+ Formula B
```

Changing the .INI Setting

The .INI setting for Risk Manager is located in the ofsrm.ini file. The entry is as follows (shown with defaults):

```
[Parallel]
SharedMemory = 16384
```

The size is in kilobytes.

Determining Shared Resource Requirements

This section describes the use of Semaphores and Shared Memory Segments by application type. In each case, the shared resources are only used while the application is running. Using the following information and knowing how OFSA is run, you can determine the total system resources that need to be available at a given time. Shared memory segments of minimum size are defined as being less than 100KB.

Application	Shared Memory	Shared Memory Sizes	Semaphores
Balance & Control	None		0
Performance Analyzer	2	1: Minimal 1: 280 * BufSize[1]	5
Risk Manager (Scenario-based run)	2	1: 1K 1: SharedMemory[#]	4 * NumProcesses[#]+ NUMROLLINTO + 9
Risk Manager (Stochastic run)	3	1: 1K 1: SharedMemory[#]	6
Transfer Pricing (No Ledger)	0		0
Transfer Pricing (with Ledger_Stat migration)	3 + SegCount[#]	2: Minimal 1: 280 * BufSize[#]SegCount[#]: 108 * SegSize[#]	10

Application	Shared Memory	Shared Memory Sizes	Semaphores
Transfer Pricing (Option Cost - Historical)	2	1: 5.7MB 2: 1088 + 640 * NumTables + (384 * NumTables + 72) * NumProcesses	2 + NumProcesses
Transfer Pricing (Option Cost - Remaining Term)	1	5.7MB	1
Transformation	0	<blank>	0

[1] See the following table for additional information.

The NumTables variable is the number of Instrument tables processed in the job.

The NumProcesses variable is the number of processes value for the job for more information).

The variables in Shared Memory, Shared Memory Sizes and Semaphores identified by the pound symbol (#) are found in the application-specific .INI file. They are described in greater detail in the following table.

Variable	.INI File Section	Description
BufSize	Ledger_Stat	Size of Ledger_Stat buffer
SharedMemory	Parallel	Size of Risk Manager shared memory Seg. If this is not a power of two, use the next higher power of two for calculations involving this variable.
SegCount	Migration	Segments to use for Ledger_Stat migration
SegSize	Migration	Size of each Ledger_Stat migration segment

Adjusting UNIX Kernel Parameters

This section describes the UNIX kernel parameters that may need to be adjusted to run the OFSA group of applications. Refer to the UNIX documentation to find out how to change these parameters as well as additional information on their meanings.

The parameters listed here are directly related to multiprocessing issues. In rare cases, other parameters may also need to be adjusted for OFSA to operate properly.

Note: The IBM AIX O/S dynamically allocates the semaphore, shared memory and other process resource kernel parameters, based on the process and system requirements. Because these parameters for IBM AIX are not directly specified by the administrator, they are not listed in this section.

Effect of Changing Kernel Parameters

Kernel parameter settings can affect performance. If they are set too small, the OFSA group of applications will not work. However, if they are set unreasonably large (for example, all parameters are set to the maximum value), your system's resources can be jeopardized. In general, do not use kernel parameter setting as the primary method of tuning the performance of your server.

Parameters Affecting System-wide Resources

The parameters listed in this section control the total number of semaphores and shared memory segments allowed to exist at any given time in the system. If any of these parameters has excessive values, the kernel may use an inordinate amount of memory. Parameter settings that are too small limit the total number of OFS applications that can be running at any point in time.

Parameters in this section may also be affected by the requirements of third party applications and the database backend. You should take these variables into account when determining the appropriate values.

Shmmni (HPUX, Compaq), shmsys:shminfo_shmmni (Sun)

The number of shared memory segments allowed to exist in the system at a given time. For OFSA, this number should be equal to the maximum number of shared memory segments that may be extant at a given moment in time.

Semmni (HPUX, Compaq), semsys:seminfo_semmni (Sun)

The maximum number of semaphore identifiers allowed in the system at a given time. For OFSA, this number should be equal to the maximum number of semaphores that are needed concurrently.

Semmns (HPUX), semsys:seminfo_semmns (Sun)

The maximum number of semaphore sets permitted. For OFSA, this should be at least the value of SEMMNI (OFSA uses one semaphore per set).

Nproc (HPUX, Compaq), max_nprocs (Sun)

Total maximum processes allowed in the system. A value of at least 1024 is recommended.

Maxuprc (HPUX, Sun, Compaq)

Total maximum processes allowed per user in the system. A value of at least 256 is recommended.

Parameters Affecting Per-process Resources

The parameters listed in this section affect the number of resources available to a single UNIX process. In general, they are set below the maximum value to prevent a single process from using an excessive amount of memory. In many situations it is acceptable to set these to their maximum legal values.

Parameters in this section may also be affected by the requirements of third party applications and the database. You should take these variables into account when determining the appropriate values.

Refer to the operating system documentation for a detailed description of the following parameters.

SVMMLIM

This parameter should be set to the maximum allowable value, usually 0x7FFFFFFF.

HDATLIM

This parameter controls the amount of memory a UNIX process can address.

The setting should be 32MB plus the size of the Risk Manager shared memory segment. Oracle installations frequently have these set to the maximum.

Shmseg (HPUX, Compaq), shmsys:shminfo_shmseg (Sun)

This parameter controls the maximum number of shared memory segments that can be used by a process.

Shmmax (HPUX, Compaq), shmsys:shminfo_shmmax (Sun)

This parameter controls the maximum size of a shared memory segment (in bytes).

Shmsys:shminfo_shmmin (Sun)

This parameter sets the minimum size of a shared memory segment. OFSA assumes it can allocate a shared memory segment as small as 1024 bytes. If you set this parameter to anything larger than 1024 bytes OFSA will not function properly.

Determining Application-Specific Memory Requirements

This section provides a set of formulas to estimate the memory requirements of the OFSA group of applications.

Total Memory Requirements for the OFSA Group of Applications

Use the following formula to calculate the total memory requirements of the OFSA group of applications:

+	Database memory requirements (determined after tuning your system)
+	Balance & Control memory requirements * the number of concurrent Balance & Control processes
+	Performance Analyzer memory requirements * the number of concurrent Performance Analyzer processes
+	Risk Manager memory requirements * the number of concurrent Risk Manager processes
+	Transfer Pricing memory requirements * the number of concurrent Transfer Pricing processes
+	Transformation Engine
=	Total memory required for OFSA

The following sections provide detailed information on calculating the parameters included in this formula.

Memory Requirements for Balance & Control

The Balance & Control memory requirements consist of the size of the program. The data memory requirements are negligible and are included in the size of the program.

Memory Requirements for Performance Analyzer

Refer to the 4.5 Product Release Notes for Performance Analyzer memory requirements.

Memory Requirements for Transfer Pricing

Use the following formula to find the total memory required for Transfer Pricing:

+	The size of the program
+	Transfer Pricing ID
+	Prepayment ID
+	Historical Rates ID
+	Cash Flow Processing structures
+	Ledger_Stat buffer size
+	Monte Carlo Rate Generator
+	The size of the data migration array
=	Total memory required for Transfer Pricing

The following formulas pertain to the three memory intensive IDs and the cash flow engine (used when a cash flow method is selected). Other IDs are used within Transfer Pricing, however, these memory requirements are negligible. The Prepayment ID and Cash Flow structures may or may not be used with every Transfer Pricing ID.

Transfer Pricing ID

Calculate the memory requirements for the Transfer Pricing ID using the following formula:

```
(n leaves * record_length)
```

where

```
record_length = 72 unpaired accounts methodology
```

and

```
record_length = 86 priced accounts methodology
```

Note: This size differs between those account methodologies that are priced and those that are unpaired.

Prepayment ID

Calculate the memory requirements for the Prepayment ID using the following formula:

```
(n leaves * y tiers per leaf * record_length)
```

where

```
record_length = 58
```

Historical Rates ID

Calculate the memory requirements for the Historical Rates ID using the following formula:

```
2 * (n IRCs * d Dates points * (t Term Points * 8 + 32))
```

Cash Flow Processing Structures

The memory used by the cash flow calculations equals the product of the number of cash flow events (payment, repricing, maturity, and so forth), the number of financial elements and the size of the data structure where the information from each row will be stored.

Use the following formula to calculate the memory required for the cash flow calculations:

```
(n events * e financial elements * record_length)
```

where

```
record_length = 408
```

```
number of financial elements = 5 for normal processing runs and 8 for
processing runs which have the detailed cash flows audit option activated
```

Note: The maximum number of structures (events * financial elements) that can be modeled is 16,000.

Option Cost Calculations

Option Cost calculations use additional cash flow structures to calculate the required memory. When running these, multiply the results of the previous formula by number of scenarios +1.

Ledger_Stat Buffer Size

To find the Ledger_Stat buffer size, multiply the BufSize setting in the ofstp.ini file under the [Ledger_Stat] section by 272.

Monte Carlo Rate Generator

The Monte Carlo Rate Generator uses two chunks of memory. One is required for low discrepancy runs and is allocated in shared memory for multiprocessing runs and in the conventional memory for single process runs. The size required is ~5.7MB in both cases. Another chunk is always allocated in conventional memory and its size in bytes is 5782 * number of scenarios.

Size of Data Migration Array

To find the total size of the migration buffer array, multiply the SegSize setting in the ofstp.ini file under the [Migration] section by 108.

Memory Requirements for Risk Manager

The following formula calculates the memory requirements for Risk Manager:

+	Shared memory
+	Memory used by IDs selected in the Processing ID
+	10,000,000 for scenario-based runs
or	
	16,000,000 + s scenarios * 5,800 for Stochastic runs) *
+	NUMPROCS + NUMSCEN * NUMEVENTS * 20 * 8
=	Total memory required for Risk Manager

The memory requirements for the specific IDs used in Risk Manager are described in the following list. IDs with negligible memory requirements are not included in this list.

Prepayment ID

Calculate the memory requirements for the Prepayment ID using the following formula:

```
(n leaves * y tiers per leaf * record_length)
```

where

```
record_length = 58
```

Discount Rate ID

Calculate the memory requirements for the Discount Rate ID using the following formula:

```
(n leaves * record_length)
```

where

```
record_length = 56
```

Forecast Balance ID

Calculate the memory requirements for the Forecast Balance ID using the following formula:

```
(n leaves * b buckets ranges * r rate levels * record_length1) + r rollinto
leaves * c contributing leaves * b buckets (average) * record_length2)
```

where

```
record_length1 = 36
record_length2 = 40
```

Maturity Strategy ID

Calculate the memory requirements for the Maturity Strategy ID using the following formula:

```
(n leaves * b buckets * record_length)
```

where

```
record_length = 24
```

Pricing Margin ID

Calculate the memory requirements for the Pricing Margin ID using the following formula:

```
(n leaves * record_length)
```

where

```
record_length = 48
```

Transaction Strategy ID

Calculate the memory requirements for the Transaction Strategy ID using the following formula:

```
(n leaves * record_length * r records per leaf (average))
```

where

```
record_length = 800
```

Leaf Characteristics ID

Calculate the memory requirements for the Leaf Characteristics ID using the following formula:

```
(n leaves * record_length)
```

where

```
record_length = 800
```

Formula Leaves ID

Calculate the memory requirements for the Formula Leaves ID using the following formula:

```
(n leaves * b buckets * f number of formulas per leaf (average) * 1024)
```

Forecast Rates ID

Calculate the memory requirements for the Forecast Rates ID using the following formula:

```
2 * (n IRCs * s scenarios * b buckets * (t terms (average) * 8 + 32)
```

Memory Requirements for the Transformation Engine

The following formulas calculate the memory requirements for Budgeting & Planning transformation, Ledger transformation, and Risk Manager transformation.

Ledger Transformation

The memory requirements for the Ledger transformation follow.

+	14,580K
+	NumProcesses * (15,490K)
=	Total memory required for the Transformation engine for Ledger

Risk Manager Transformation

The memory requirements for the Risk Manager transformation follow.

+	19,800K
+	NumProcesses * (19,820K)
=	Total memory required for the Transformation engine for Risk Manager

Configuring the Request Queue Log File

The following two settings are associated with configuring the Request Queue (RQ) log file.

Setting	Description
MaxFileSize	This setting specifies the maximum allowable size of the RQ log file. The maximum file size is equal to this value multiplied by 1024 (the number of 1024 byte blocks). The default setting is 512. A setting of 0 (zero) enables the file to grow without limits.
IdealFileSize	This setting specifies the ideal size for the RQ log file. The maximum file size is equal to this value multiplied by 1024 (the number of 1024 byte blocks). When the log file reaches maximum size, the data contained in the log file is flushed out until the ideal file size is reached. The default setting is 256.

The following sections discuss the environment variable settings for the RQ log file for each of the servers included in this chapter.

Sun Environment Variables

The rq command script sets the following environment variables for Sun servers.

Variable	Description
LD_LIBRARY_PATH	This is the path used to find shared object libraries. During the initialization phase of a process startup, the shared object libraries are dynamically linked into the process.
INIPATH	This is the path to the initialization files and other files read by OFSA executables. The path is set to the /OFSA_INSTALL/etc directory under the OFSA installation directory.
NLSPATH	Contains the path to search for the OFSA message catalogs. In addition to what is already defined for the system, it should include **OFSA_INSTALL/etc/nls/msg/%L/%N:OFSA_INSTALL /etc/nls/msg/C/%N** where OFSA_INSTALL is the installation directory for the OFSA group of applications. This path causes it to search the current language, then to search the default C locale, which is identical to the en_US locale.
LANG	Should be set to the desired language. C and en_US (U.S. English) are currently supported.

HP-UX Environment Variables

The rq command script sets the following environment variables for HP- UX servers.

Variable	Description
SHLIB_PATH	This is the path used to find shared libraries. During the initialization phase of a process startup, the shared libraries are dynamically linked into the process.
INIPATH	This is the path to the initialization files and other files read by OFSA executables. The path is set to the /.../OFSA_ INSTALL/etc directory.

Variable	Description
NLSPATH	Contains the path to search for the OFSA message catalogs. In addition to what is already defined for the system, it should include **OFSA_INSTALL/etc/nls/msg/%L/%N:OFSA_INSTALL /etc/nls/msg/C/%N** where OFSA_INSTALL is the installation directory for the OFSA group of applications. This path causes it to search the current language, then to search the default C locale, which is identical to the en_US locale.
LANG	Should be set to the desired language. C and en_US (U.S. English) are currently supported.

IBM-AIX Environment Variables

The rq command script sets the following environment variables for IBM servers running AIX:

The following table lists and describes each of these variables.

Variable	Description
LIBPATH	This is the path used to find shared libraries. During the initialization phase of a process startup, the shared libraries are dynamically linked into the process.
INIPATH	This is the path to the initialization files and other files read by OFSA executables. The path is set to the /.../ofsa/etc directory.
NLSPATH	Contains the path to search for the OFSA message catalogs. In addition to what is already defined for the system, it should include: OFSA_INSTALL/etc/nls/msg/%L/%N:OFSA_ INSTALL /etc/nls/msg/C/%N where OFSA_INSTALL is the installation directory for the OFSA group of applications. This path causes it to search the current language, then to search the default C locale, which is identical to the en_US locale.
LANG	Should be set to the desired language. C and en_US (U.S. English) are currently supported.

Other Configuration Issues

This section discusses the following, additional configuration issues:

- Capturing SQL for database optimization
- Core files in Request Queue
- Cleaning shared resources
- Running multiple software instances

Capturing SQL for Database Optimization

This section describes how the SQL that OFSA uses for processing can be captured for database optimization.

> **Caution:** Careless use of this facility can result in the use of a large amount of disk space and performance degradation because of the logging of extraneous information.

.INI [debug] Section of the Application-specific .INI Files

The following table provides useful values for the various output logs that you can define in the [debug] section of the application-specific .INI files.

Item	Value	Meaning
FILENAME	name	Name of log file, saved as <FileName>.<job_number>.log.
CALC_LOG	1	Shows significant calculation SQL in log file.
	2	Shows units of work as processed in log file in addition to significant calculation SQL.
ERROR_LOG	1	Places error output in log file.
	3	In addition to OFSA error output, includes the errno of failed system calls in the log file. This is useful for diagnosing kernel parameter issues involving semaphores and shared memory segments.
ACCESS_LOG	2	Shows all SQL.

Setting any of the log levels to 0 [zero] turns off output for the given log.

To view meaningful SQL for database optimization, set CALC_LOG to a value of 1.

Caution: Turning on the logging functionality in the ofs.ini file results in RQ outputting log information. If RQ is left running for any significant length of time with logging enabled, the log files can grow quite large. This is different from the log file enabled on the command line.

Oracle does not warrant that the SQL used by OFSA will remain consistent across releases, including incremental versions of the same release. Also, the SQL generated by OFSA may change, based upon user modifications to IDs and differences in instrument data from time period to time period.

Core Files

Occasionally core files from OFSA processes end up in the RQ working directory (usually OFSA_INSTALL_dir/log where OFSA_INSTALL_dir is the installation location of OFSA) due to situations that are beyond the control of the application, such as running out of system resources. These files are named either core.<job_number> or core.<job_number>.<process_number> where <job_number> is the job number assigned to the process by OFSA and <process_number> is a subprocess number assigned by OFSA.

If the file name is core.<job_number> the primary process has dumped core. If the file name is core.<job_number>.<process_number> both multiprocessing and subprocesses have dumped core. In either case the core is reported in the RQ log file.

In the event that two OFSA processes dump core simultaneously, the core file may not correspond to the process as reported in RQ.

Generally, you can delete core files. However, in some instances, when a Technical Analyst in Oracle Support Services cannot reproduce the problem, you may be requested to send the core file to Oracle Support Services.

Cleaning Shared Resources

When an OFSA process using shared memory abnormally terminates, the shared memory and semaphores that it was using will not be cleaned up correctly and will persist until either the server is rebooted or you clean up the resources. Shared

resources can be cleaned up by running the **rmipc** command located in the bin directory under the OFSA_INSTALL directory.

Note the following precautions before you run the rmipc command:

- OFSA processes should not be running when this command is run.
- OFSA should not be sharing a uid with anything else, including Oracle database processes.
- Users that run OFSA and Oracle under the same uid must shut down Oracle before using this command.
- The command must be run using the same user as the one used for OFSA.

The rmipc command operates by deleting all shared resources owned by the uid that runs the command. This includes semaphores, shared memory segments and message queues.

Caution: If the OFSA application are running under the same uid as non-OFSA applications, the use of this command can result in unintended consequences.

Multiple OFSA Server-Centric Application Instances

A server can be set up to run more than one instance of the OFSA application as long as each instance of the application is running against a different OFSA database. Each RQ is started and run independently from the other and should specify a different database instance and log file. Different databases can be configured to use separate .INI files by copying and editing the rq shell script located in the bin directory for each database release.

7

Client Software Installation and Configuration

This chapter provides information on installing, configuring, and upgrading the client-side of the Oracle Financial Services Applications (OFSA) group of applications. The following topics are included in this chapter:

- Verifying the Correct Client Workstation Software
- Installing the Client-Side OFSA Software
- Upgrading the Client-Side Software
- Setting the Date Format in NT 4.0
- Troubleshooting Client Installations
- Running Multiple OFSA Applications Simultaneously
- .INI Settings
- Debug Settings
- Application-Specific Settings
- Client PC Memory Considerations

Verifying the Correct Client Workstation Software

This section provides information on the software that needs to be in place before you begin installing the OFSA software and illustrates the communication layers between the client and database.

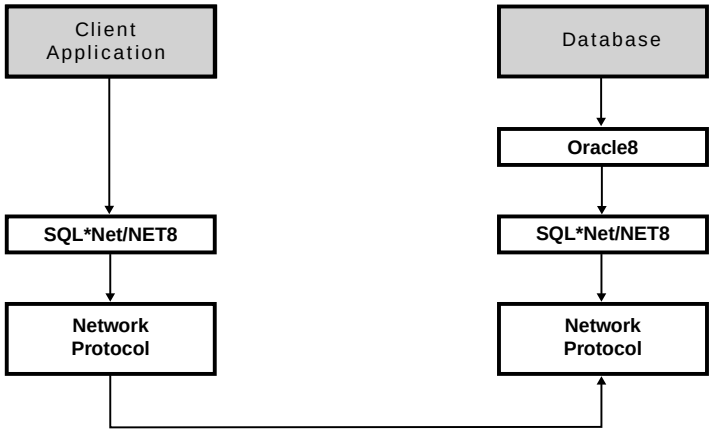
Verifying the Installation of the Client Workstation Environment

Before you begin installing this release of the client-side software make sure that either Windows 95/98 or NT is installed on the client workstation. Windows 3.1 is not supported.

Verifying the Installation of Your Network Protocol

Verify that your network protocol is set up and configured correctly.

The following illustration depicts the layers of communication required for the OFSA group of applications.



Installing the Client-Side OFSA Software

The client-side of the OFSA software is shipped on CD-ROM and uses Oracle Installer for the installation routine.

Note: The Market Manager application is not included on the OFSA 4.5 CD. However, Market Manager version 4.0 is compatible with the FDM 4.5 database (with the Market Manager database objects installed). Use the OFSA 4.0 CD to install this version Market Manager. Refer to the 4.0 version of the *Oracle Financial Services Installation and Configuration Guide* for information regarding how to install Market Manager.

Note: Installation for Oracle Budgeting & Planning is described in Chapter 7, "Client Software Installation and Configuration".

Installing on Windows 95/98

Oracle recommends that you always perform a reboot prior to installing OFSA on a Windows 95 or Windows 98 PC. This clears any potential OFSA DLLs that might be running in memory. If you do not perform a reboot, the OFSA installation can terminate abnormally if existing OFSA DLLs are running in memory.

16-bit and 32-bit Installations

OFSA applications and functionality use both 16-bit and 32-bit technology components. To properly install the required technology you need to run the installation routine twice, first for the 16-bit components and then for the 32-bit components.

Each set of components is located in different directories on the CD-ROM. Select the Windows directory to install the 16-bit components and the WIN32 directory to install the 32-bit components.

Required Technology Components

Each OFSA application in the Oracle Installer requires additional technology components for operation. All required technology components must be installed in order for the application to function properly.

The required technology components for OFSA are categorized as follows:

- 16-bit components included in the Oracle Installer
- 32-bit components included in the Oracle Installer
- Third-party components

Third-party components are defined as required technology components that are not included in the Oracle Installer. Non-Oracle and Oracle applications (such as Oracle Discoverer) not included on the OFSA CD are categorized as *Third-party*.

Refer to Chapter 3, "Certifications" for a detailed list of components (and certified versions) required for the OFSA applications. When installing an OFSA application, all 32-bit required components included with the Oracle Installer are installed automatically. This means that you select only the specific OFSA application to install and all of the 32-bit required technology components are automatically installed as well.

However, 16-bit required technology components must be installed separately from the Windows directory on the CD. All third-party components must also be installed separately. Third party components are marked with an asterisk (*) in the Certifications chapter.

Note: Be sure to install Discoverer 3.1. OFSA requires Discoverer for the OFSA Standard Reports.

Note: Technology stack references for Market Manager version 4.0 are not included in the Certifications chapter. However, Market Manager version 4.0.3 is compatible with the FDM 4.5 database.

Installing the Software

This section provides the steps for installing the OFSA applications and both 16-bit and 32-bit components from the CD-ROM.

Installing the 16-bit Components

Only 16-bit technology components are installed using this routine. OFSA applications and functionality are installed with the 32-bit installation routine.

Complete the following steps to install the 16-bit technology components.

1. Insert the CD-ROM into the CD-ROM drive.

   Caution: You can run Oracle Installer only from a local drive or a drive that has been mapped using Windows Explorer (that is, a letter assignment exists for the drive). Do not use Network Neighborhood to run Oracle Installer. If you do, it does not install the OFSA applications properly.

2. Select the Windows directory and run setup.exe.

   This launches Oracle Installer and leads you through the following sequence of dialogs.

   a. Language dialog

      Use the selection box to select the language for Oracle Installer. The default setting is English.

   b. Oracle Installation Settings dialog

      The information requested in this dialog includes your company name and the Oracle home.

      The default setting for the 16-bit components, for both NT and Windows 95/98, is C:\ORAWIN.

      Caution: The Oracle Installer does not support spaces or special characters in target directory names. Use dash (-) or underscore (_) instead when specifying an installation target directory name.

   After selecting the directory location for the OFSA software, the Software Asset Manager dialog appears.

3. From the Software Asset Manager dialog select the technology components you want to install.

   Components available for installation appear in the left pane of the dialog. Oracle components currently on the client, in the Oracle Home directory, appear in the right pane.

a. If you are installing a single component, use your mouse to highlight that application.

b. If you are installing multiple components, use your mouse and either the Shift or Control key to highlight the desired items.

c. After highlighting the components you want, click Install.

If either configuration or software version conflicts exist, dialogs appear. Resolve these as recommended in the dialogs.

4. When the installation is complete the Installation Completed dialog appears. Click OK.

5. The Software Asset Manager dialog re-appears. Click Exit.

This completes the installation of the 16-bit components.

Feedback on Space Requirements and Selected Technology Components

Two information boxes appear below the left and right panes of the Software Asset Manager dialog. The first box provides feedback on space requirements for the OFSA items selected from the left pane and space availability on the client. The second box lists the technology components you are installing on the client.

Installing the 32-bit Components and OFSA Applications

OFSA applications and functionality as well as 32-bit technology components are installed using this installation routine.

Although Oracle Installer enables you to install multiple MAJOR releases (differing in first two digits, e.g. 4.0 and 4.5) of the OFSA software on the client, this option is not recommended. You should always uninstall existing OFSA software before installing a new version of the same major release (such as 4.5 and 4.5.1).

Complete the following steps to install the OFSA applications, functionality and technology components.

1. Insert the CD-ROM into the CD-ROM drive.

Caution: You can run Oracle Installer only from a local drive or a drive that has been mapped using Windows Explorer (a letter assignment exists for the drive). Do not use Network Neighborhood to run Oracle Installer. If you do, it does not install the OFSA software properly.

2. Select the WIN32 directory and run setup.exe.

   This launches Oracle Installer and leads you through the following sequence of dialogs.

   a. Oracle Installation Settings dialog

      Use this dialog to enter your company's name, the location of the ORACLE_HOME directory and the language setting.

      The default setting for ORACLE_HOME for NT is C:\ORANT and for Windows 95/98 is C:\ORAWIN95.

      Caution: The Oracle Installer does not support spaces or special characters in target directory names. Use dash (-) or underscore (_) instead when specifying an installation target directory name.

      The default language setting is English.

   After selecting the directory location and language setting the Software Asset Manager dialog appears.

3. From the Software Asset Manager dialog, select the application or applications and appropriate technology components you want to install. Applications and components available for installation appear in the left pane of the dialog. Oracle applications and components currently on the client, in the Oracle Home directory, appear in the right pane.

   a. If you are installing a single application or component, use your mouse to highlight that application or component.

   b. If you are installing multiple applications and components, use your mouse and either the Shift or Control key to highlight the desired items.

      Note: The installation of Budgeting & Planning requires additional steps. If you are installing Budgeting & Planning, see Chapter 8, "Budgeting & Planning Server-Side Installation and Setup".

Note: A + precedes some items in the left pane of the Asset Manager window. This indicates that a sub-hierarchy of selections are available. Access these additional selections by double-clicking on the selection that is preceded by the +.

A - indicates that all available selections at the sub-hierarchy level are shown.

4. After highlighting the applications and appropriate technology components, click Install.
5. The Oracle Financial Services Product Home dialog appears.

   This dialog identifies the directory location where you are installing the OFSA software.

   The default setting for NT is C:\ORANT\OFSA45.

   The default setting for Windows 95/98 is C:\ORAWIN95\OFSA45.

Note: All Oracle applications must be installed in an Oracle Home directory.

Caution: The Oracle Installer does not support spaces or special characters in target directory names. Use dash (-) or underscore (_) instead when specifying an installation target directory name.

6. After selecting the directory location the installation is launched and technical dependencies analyzed.

   The installation routine begins comparing versions of technical components on the client against versions on the CD-ROM. If an older version exists on the client, an Update dialog appears, prompting you to upgrade to the newer version. Always click OK to accept the newer version.
7. After all dependencies have been analyzed and updated, the Required Products dialog appears. This dialog lists all required technical components to be

installed and any additional applications and components that you have selected. Click OK.

8. When the installation is complete the General Information dialog appears indicating the installation was successful. Click OK.

9. The Software Asset Manager dialog re-appears. Click Exit.

   This completes the installation procedure for the OFSA applications and functionality and 32-bit technology components.

Installing Discoverer with FDM Administration

Caution: Installing Discoverer after installing the FDM Administration application causes a DLL error.

Always install Discover before installing the FDM Administration application. If you install Discoverer after installing the FDM Administration application, you must re-install FDM Administration for it to work properly.

Installing Budgeting & Planning

Because Budgeting & Planning works in conjunction with Oracle Financial Analyzer (OFA), additional steps are required to successfully complete an installation of this application. Refer to Chapter 8, "Budgeting & Planning Server-Side Installation and Setup" for additional instructions on installing this application.

Installing Discoverer Integrator

Only administrators or power users responsible for integrating Oracle Discoverer for the OFSA Reporting Data Mart should install the Discoverer Integrator.

Caution: If you install Discoverer with the OFSA group of applications and, at a later time, decide to uninstall National Language Support (NLS), which is included in OFSA, Discoverer no longer works.

If you proceed to uninstall NLS, a dialog box appears alerting you that the removal of the NLS component will cause other Oracle applications to fail. It is recommended that you keep NLS as a component of OFSA.

Installing FDM Administration

Only administrators responsible for managing security and objects within the database should install the FDM Administration application.

Installing the Documentation HTML Help Files

The Oracle Installer automatically installs documentation for the OFSA group of applications whenever you install OFSA applications. You can also install the documentation separately.

To install the Documentation HTML Help files separately, highlight Oracle Financial Services Documentation 4.5 in the left pane and click the Install button. During the installation process, the Documentation Install Options dialog box appears, prompting you to select one of the three following options:

Option	Description
Install documentation on local disk	Select this option if you want to copy the HTML Help files to the disk drive on the local computer.
Use documentation on disk server or CD-ROM	Select this option if you plan to access the HTML files from the CD-ROM in the CD-ROM drive rather than copying the files to the local computer.
Use documentation on web server	This option enables you to view the HTML files through your browser, using a URL to access the files.

After selecting the installation method, you are prompted to designate the source drive location of the documentation files. If you are installing the HTML files on the local disk or using the CD-ROM, enter the drive information and click OK.

If you select the web server option, be sure that the documentation HTML files have been installed on your web server before completing this procedure. You also need to know the URL to access the HTML files. Contact your System Administrator if you are unsure about the installation of these files on the web server or the URL.

The Documentation Install Options dialog box provides additional information about each option through the Help button. To access this information, select the installation method you are considering and then select Help. A description of the selected method appears.

Software Required for HTML Help Files

In order to operate Help from within OFSA, Internet Explorer version 4.0 or greater must be installed. Then, you can use the browser of your choice to access OFSA Help.

Browser Support for HTML Help Files

All browser versions are supported in NT.

The following browser versions are supported in Windows 95/98:

Browser	Version
Netscape	Version 4.x
Internet Explorer	Version 4.0.x with the following exception: 4.40.308

Configuring SQL*Net and Oracle NET8

Oracle Installer automatically installs or verifies the prior installation of the networking software required by the applications you are installing on the client. There are three different networking applications for communication with the OFSA database:

- 16-bit SQL*Net

- 32-bit SQL*Net
- Net8

> **Note:** References for Market Manager version 4.0 are included here even though Market Manager is not part of the OFSA 4.5 CD. Market Manager version 4.0 is compatible with the FDM 4.5 database and is therefore listed for reference purposes.

The following table lists the appropriate networking software for each OFSA application / functionality:

OFSA Application/Functionality	Oracle Networking Software
Balance & Control	16-bit SQL*Net
Budgeting & Planning	None - Web enabled
Discoverer	Net8
Discoverer Integrator	Net8
FDM Administration	Net8
Market Manager (version 4.0)	Net8
	32-bit SQL*Net
Performance Analyzer	16-bit SQL*Net
Portfolio Analyzer	16-bit SQL*Net
Rate Manager	Net8
Risk Manager	16-bit SQL*Net
Transfer Pricing	16-bit SQL*Net
Transfer Pricing, Risk Manager and Performance Analyzer knowledge engines	Net8

The following table provides information needed to configure both SQL*Net and NET8.

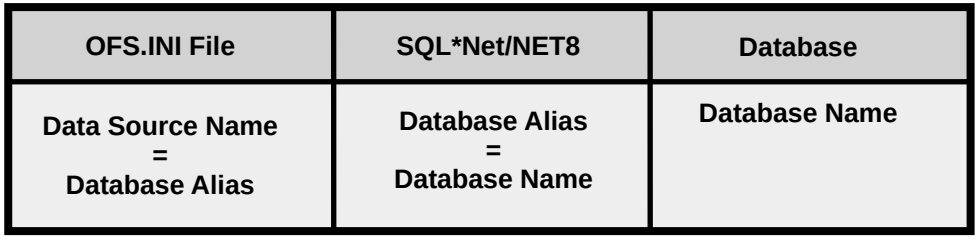
Required Information	Description
Database alias	This is the name by which the networking software recognizes the database. You also need this name to configure the OFS.INI file.
Hostname[1]	This is the name of the server on which the database is located.
Database instance‡	This is the name by which the server recognizes the database.

[1] Your DBA needs to provide the names of the hostname and database instance. You should have these names prior to configuring the networking software.

Establishing the Link Between the Client Workstation and the Database

Both SQL*Net and NET8 give you the option of providing one name for the database instance on the server and a different name (database source) for the same database as it is recognized by the OFSA group of application. SQL*Net and NET8 provide the link through the database alias.

The following diagram illustrates the link between the client workstation and the server, through SQL*Net or NET8, to access the database.



Configuring ODBC

The following table provides information to configure the ODBC drivers.

OFS Application/Functionality	Datasource Requirements	.INI File Configuration
Import/Export functionality	dBase IV	■ The datasource name = xbase.

OFS Application/Functionality	Datasource Requirements	.INI File Configuration
Market Manager (version 4.0)	32-bit Oracle datasource	■ The database alias is the name used by SQL*Net / NET8. ■ The datasource name is the name of the database as it appears in the OFSA application.

Modifying the OFS.INI File

The installation program creates the OFS.INI file and locates it in the directory where your operating system resides. Typically, this is in the WINNT or WIN95 directory. This file provides a list of datasources available within the OFSA group of applications and data unique to each datasource.

The installation program creates a sample OFS.INI file. You can modify this sample file or create your own.

Note: The OFS.INI file is not used for FDM Administration, Rate Manager, or Market Manager.

Data Sources

An example of a the Data Source section of the OFS.INI file appears as follows:

```
[DataSources]
Oracle_Example          = YES
[Oracle_Example]
DriverType              = ORACLE
Database                = PrimaryDatabase
LogonID                 = USER_ID
```

The following table describes each of the components of the Data Source section in OFS.INI file.

OFS.INI Component	Description
[Datasources]	This is the list of datasources (databases) created within the OFSA group of applications. In this example, the datasource is called *Oracle_Example*. This is the database name that appears in the OFSA applications. If you type YES, this datasource appears on the login window when the application is launched. If you enter NO, this datasource does not appear. You can list multiple datasources in the OFS.INI file.
[Oracle_Example]	This heading indicates that the information entered pertains to this datasource only. The name of this database (Oracle_Example) appears in the list of Datasources. If you are using multiple databases, each needs to be entered under [Datasources] as shown by the example in which Oracle_Example is entered as a database within the OFS applications.
DriverType	This indicates the database driver to use to connect to the datasource. This should always be ORACLE and is not case-sensitive.
ServerName	Enter the database alias here. This is the database name that SQL*Net or NET8 will recognize. See the section entitled "Establishing the Link Between the Client Workstation and the Database" in this chapter for more information.
Database	Normally this is the same as ServerName. If ServerName is blank, this should be the value of the $ORACLE_SID.
Login ID	This is the default login ID for the end user (user ID). The login ID entered here appears on the Login window.

Tree Rollup Save Behavior Settings

A configuration option controls integrity checking of IDs when a Tree Rollup is saved. Specifically, when the levels of a Tree Rollup are inserted or deleted dependent Allocation IDs, Table IDs, and Tree Filter IDs reflect the Tree Rollup change. A record of the change is reflected in OFSA_MESSAGE_LOG. A standard Discoverer report, errormsg.dis, can be installed to navigate OFSA_MESSAGE_ LOG. The syntax of the configuration option in OFS.INI is:

[options]

checkallocinteg=1

where

checkallocinteg=1 means check integrity

checkallocinteg=0 means do not check integrity

If the configuration option does not exist in OFS.INI, the default is 0, do not check integrity.

Request Queue Communication Settings

The [OFSRQ] section of the OFS.INI file gives you control over how often the Server Status window is updated and how long to wait before determining that the Server Request Queue component is not available.

ClientPollInterval

This setting controls how often the Server Status window is refreshed. Time is measured in milliseconds. The default setting is 1,650 milliseconds (1.6 seconds).

ServerTimeOut

OFSRQ	Options	Default Setting
`ClientPollInterval = <number>`	Time in milliseconds	Default = 1650

This setting controls how long to wait for a response to a ping request. The time is measured in seconds. The default setting is 30 seconds. This setting should be at least 10 times the poll interval used by Request Queue (default is 3 seconds). If the ServerTimeOut setting is too short, false timeouts can occur. For more information on this setting refer to Chapter 20, "Request Queue".

OFSRQ	Options	Default Setting
`ServerTimeOut = <number>`	Timeout in seconds	Default = 30

Upgrading the Client-Side Software

Before running the upgrade process uninstall your existing OFSA software. Highlight the previous version (OFSA 3.5 or OFSA 4.0) of the OFSA applications within the Oracle Installer and select Remove. This removes the components from your system. You can then proceed with installing the OFSA 4.5 applications.

Setting the Date Format in NT 4.0

If the date format does not appear correctly in the OFSA applications and you are running NT 4.0, add the following key to your NT registry in HKEY_LOCAL_MACHINE /SOFTWARE/ORACLE:

NLS_DATE_FORMAT:REG_EXPAND_SZ:MM/DD/YYYY

By adding this key, you ensure that NT-based Oracle applications you are running use the OFSA-required date format when interpreting DATE literals.

Troubleshooting Client Installations

General Guidelines

As a general rule, always un-install any existing installation of OFSA on the client PC using the Oracle Installer prior to installing a new version. However, in some situations, you can encounter difficulty in successfully un-installing OFSA using the Oracle Installer due to file or registry corruption. In these situations, refer to the following guidelines:

- Close Windows Explorer before installing or de-installing OFSA software.
- Install the OFSA application individually (overwriting the existing installation). This corrects any registry errors for the individual application. Once the application has been re-installed, remove the application using the Oracle Installer. The application can now be installed cleanly.
- If the guideline does not correct the problem, install the Oracle Financial Services Applications Clients 4.0 entry in the Oracle Installer. This installs all of the OFSA applications and corrects any registry errors common to all of the applications. Once the installation is complete, remove all of the OFSA applications using the Oracle Installer. Individual applications or all OFSA applications can now be installed cleanly.

Avoiding “Permission Denied” Messages for DLL Files When Installing or Deinstalling OFSA Software

This message results from DLLs in use by other programs that are running at the same time that you are installing or deinstalling OFSA applications or if a task called JAVA is running at the same time.

Sometime OFSA applications or the Net8 Easy Config program leaves a JAVA task running after the application or program has been closed. This task uses NETDIR.DLL as well as other DLLs that begin with the letter "J."

Avoid this message by taking the following precautions:

1. Close all applications, including Explorer, before beginning your installation or deinstallation.

2. Check your system for open windowless tasks, such as JAVA or MS Office Find Fast.

- In Windows NT, open the Task Manager and close all programs that are running.
- In Windows 95/98, use Ctrl-Alt-Delete to activate the Close Program function and close all tasks that are running.

> **Note:** The *Permission Denied* message can continue to appear even after closing all open programs and functions. If this occurs, reboot your system.

Avoiding an "Invalid instantiation string" Error When Installing Applications and Documentation In a Single Step on Windows 95/98

If you are installing on Windows 95/98, you cannot install both the OFSA software and the documentation HTML Help files at the same time. If you do the *Invalid instantiation string* error message appears.

Avoid this problem by using a two-step installation process. Install the OFSA software first, then install the documentation HTML Help files.

Installing the Documentation Icon Manually When It Does Not Appear in the Programs Menu

If OFSA documentation is not installed on a local PC, the icon for documentation does not appear in the Programs menu.

If you are installing the documentation HTML Help files from either a file server or a web server, include the documentation icon in your Program menu by following these steps:

1. Go to Settings>Taskbar>Start Menu Programs tab>Add button.
2. Use the browse functionality to locate the file "index.htm."

3. Place the file in the OFSA Program folder.

If you plan to access documentation HTML Help files using the client-side CD-ROM from the local drive, follow the steps but, in step 2, use the following link: win32\OFSOLH\index.htm.

Avoiding a "write error" Message When Installing or Deinstalling OFSA Software

This message can appear if a directory being installed or deinstalled by Oracle Installer is open in Windows Explorer. When this message appears, you may need to reboot your system to clear the problem.

Windows Explorer Hangs After Deinstalling OFSA Software

If Windows Explorer is open to a directory that Oracle Installer deinstalls, the Explorer program can hang. The result can be a general paralysis of your system, including the inability to look at directories or to start new programs. This situation may persist even after rebooting your system because Explorer starts in the last directory, which is now non-existent, that was open before you shut down your computer.

Fix this problem by starting Explorer from Start>Run and entering a directory in the Open: field, such as "explorer c:\".

Install Oracle Applications Desktop Integrator (ADI) First If Installing Both ADI and OFSA 4.0

If you plan to install ADI, it is essential that you install ADI first, then install OFSA 4.0. If you reverse the order and install the OFSA software first, several OFSA objects are overwritten with older ADI DLLs and executables.

The following files are overwritten (example shows an NT installation):

- C:\orant\bin\IM25W32.DLL
- C:\orant\bin\MMC21W32.DLL
- C:\orant\bin\MMI21W32.DLL
- C:\orant\bin\REGSVR32.EXE
- C:\orant\bin\UTL25W32.DLL

Running Multiple OFSA Applications Simultaneously

The Oracle Installer automatically configures the OFSA group of applications to Run in Separate Memory Space on NT client. This enables you to launch multiple instances of the individual OFS applications on your NT PC simultaneously.

Because Windows 95/98 does not support running applications in a separate memory space, it is not possible to launch multiple instances of the individual OFS applications on a Windows 95/98 PC.

.INI Settings

Descriptions of the .INI settings, found in the [options] section of the .INI file are listed in the following tables.

Maximize

This attribute defines whether the multiple document interface (MDI) windows are created and displayed in full or regular window mode.

Parameter	Selections	Default Setting
`Maximize=[0|1]`	0 - MDI windows are created in full window mode 1 - MDI windows created in regular window mode	Default = 1

FillColor

FillColor defines the background colors of the dialog controls for each application's MDI window. The content of red, green and blue is controlled by the numbers associated with each letter.

- Red is controlled by the x value
- Green is controlled by the y value
- Blue is controlled by the z value

Parameter	Selections	Default Setting
`FillColor=[xxx yyy zzz]`	xxx = 000 to 255 yyy = 000 to 255 zzz = 000 to 255	Default = 000 128 128

The default setting is teal green and represented by the value “000 128 128.” White would be represented by “000 000 000.”

Modulus

When making SQL fetches from the database, OFSA keeps a running count of the rows fetched. This incremental number can be displayed on the status bar in increments you define using the modulus option. The **n** value you select tells OFSA to update the status bar display each time the number of fetches reaches this pre-set number.

For example, if n is set at 300, OFSA displays the running total every 300 rows. At 300 rows fetched, it displays 300, at 600 row it displays 600 and so forth.

Parameter	Selections	Default Setting
`Modulus=[n]`	n = 1 to n	Default = 100

Face

Face defines the type of font used for the spreadsheet and tree display.

Parameter	Selections	Default Setting
Face = [Font_name]	Any installed True Type and raster (bitmap) fonts	Default = Arial

FaceWeight

Face weight sets the weight of the font used.

Parameter	Selections	Default Setting
FaceWeight = [n]	n = 400 (normal weight) or 800 (bold weight)	Default = 400

Size

Size defines the font size, in points, for the font defined by Face.

Parameter	Selections	Default Setting
`Size=[n]`	n = 1 to 144	Default = 16 A 72-point font is one inch high.

Italic

The italic font attribute for the font defined by the Face is turned on or off with this parameter.

Parameter	Selections	Default Setting
`Italic=[0|1]`	0 - Italic attribute not used 1 - Italic attribute used	Default = 0

Debug Settings

Each application's .INI file includes a [Debug] section. You would normally enable this only in response to a request from Oracle Support Services.

OFSA provides three debug levels, defined as Level 1, Level 2, and Level 3. However, Level 3 is the only active debugging level. Level 1 and Level 2 are either disabled or not used.

> **Note:** Debugging in this section refers only to the client and not the server.

Level 3

Used for access layer debugging. Level 3 defines the lowest and most detailed debugging level. This level produces maximum debugging information.

The OFSA applications write SQL statement syntax at this level.

Debug Level	Options	Default Setting
Level3=[0|1]	1 - Level 3 debug is ON 0- Level 3 debug is OFF	Default = 0

Level 2

Currently not used.

Level 1

Currently disabled. Level 1 defines the highest and least detailed debugging level. This level is intended to produce minimum debugging information.

Debug Level	Options	Default Setting
Level1=[0|1]	1 - Level 1 debug is ON 0- Level 1 debug is OFF	Default = 0

Application-Specific Settings

Each application has its own .INI file. Application-specific settings are described in the following sections.

Note: Some applications do not have any application-specific settings. If this is the case, they are not included in this section.

Balance & Control

Balance & Control settings are located in the OFSBC.INI file.

[Options]

Determines the maximum size of a cached looup table.

Maxlooup = 16000

If a lookup table contains more than the maxlookup rows, the rows are read when needed using a parameterized select statement.

Performance Analyzer

Performance Analyzer settings are located in OFSPA.INI.

[Options]

epm

Extended Printer Memory - the number of rows kept in memory before flushing for a report preview.

```
epm=[n]
```

n = minimum = 1, maximum determined by available memory

Default = 500

Client PC Memory Considerations

Client PCs may experience memory limitations when operating against database with a large number of Leaf Values. The FDM database supports up to 200,000 Leaf values in version 4.5. This section describes some techniques when you are running in such environments:

> **Note:** For the PCs used to work with a database containing 200,000 leaves, 128 megabytes of memory are recommended.

Tree IDs

Opening a tree type ID loads all leaves into memory. You should only open one large tree in a session. If you need to edit two or more trees, quit and restart the application between edit sessions. Within a single tree editing session, you can complete a significant number of edits and changes before your system memory is affected. Some leaves have associated leaf tables (like Org Unit and Original Org unit) and editing those leaves requires the other leaf list to load into memory. This strains the system memory.

To avoid showing too many leaves and to increase response time, when editing a tree, first collapse the entire tree. Then Search on the Tree side for the leaf or node

you want. When Search finds your leaf it opens only that branch and the related branches leading to it.

Caution: Do not use the Search and Focus feature. Search and Focus focuses down to the target leaf when it finds it and you will lose your tree context.

When creating a tree, collapse the Tree down to just the levels you need or first focus on the branch that you are editing. Then set the Display side to Orphans and perform your search by leaf value or description on the Display side. You can then drag over the leaves you find from the right side to the node on the left.

Leaf Setup

Leaf Setup was designed to minimize memory use. Make an effort to display only the leaf values and ranges that you need. If you click the Edit button to edit details about a leaf value, you will load all leaves into memory (causing a delay during this process) and that will stress your system. Limit editing of leaves to one or possibly two leaf types during one session.

Caution: If you attempt to edit too many leaf types, your system can become unstable and crash. Close the application and restart it before editing another leaf type.

8

Budgeting & Planning Server-Side Installation and Setup

This chapter presents information on the software components, in addition to the Budgeting & Planning application, that are required to run the Budgeting & Planning process and additional installation routines that are essential to completing the installation process.

The following topics are addressed in this chapter:

- Before Installing Budgeting & Planning
- Budgeting & Planning Server Installation
- Administering the Budgeting & Planning Databases
- Configuring the Web Listener and Java Client

Budgeting & Planning is designed to work in conjunction with the following Oracle products:

- Oracle Application Server 4.0.8.1
- Oracle Express Server 6.3.0.1
- Oracle Web Agent 6.3.0.1
- Oracle Financial Analyzer 6.3.0.0
- J Initiator 1.1.7.29

Note: Try to obtain the exact versions. If you are unable to order these exact versions, contact Oracle Support Services before proceeding.

If you are running the Oracle Financial Services Applications (OFSA) group of applications, install the applications mentioned above in addition to the Oracle Financial Data Manager (FDM) database.

If you are responsible for installing Oracle Express Server and Oracle Financial Analyzer (OFA), refer to the appropriate sections in the Installation Guides for these products.

You install Budgeting & Planning using Oracle Installer, which is included on the OFSA CD-ROM. Refer to Chapter 6, "UNIX Server Installation and Configuration" for additional information on installing OFSA applications and configuring your OFSA environment.

Before Installing Budgeting & Planning

Before you begin to install Budgeting & Planning you must obtain the following installation and configuration manuals.

- Oracle Application Server for Windows NT, Installation Guide, Part No. A58756-03 or Oracle Application Server for Sun SPARC Solaris 2.x, Installation Guide, Part No. A58755-03
- Oracle Express Web Products, Installation Guide, Release 6.3, Part No. A75233-01
- Oracle Express Database Administration Guide, Release 6.3, Part No. A74956-01
- Obtain the Oracle Express Server Installation and Configuration Guide Release 6.3 appropriate for your environment:
  - For Windows NT, Part No A75298-01
  - For Sun Sparc Solaris, Part No. A75299-01
  - For HP 9000 Series HP-UX, Part No. A83777-01
- Oracle Financial Analyzer, Installation and Upgrade Guide, Part No. A68143-01

Budgeting & Planning 4.5 is designed to run with:

- Oracle Application Server 4.0.8.1
- Oracle Express Server Release 6.3.0.1
- Oracle Web Agent 6.3.0.1 (with patch "OWA630_P1.exe")
- Oracle Financial Analyzer (OFA) Release 6.3.0.0 (with patch "6325_2.exe")

Install these applications in the exact order as they are listed. Install all of these products before installing Budgeting & Planning.

> **Note:** When you install the Oracle Application Server before upgrading/re-installing the Oracle Express Server, the Oracle Express Server installation automatically configures the web listener included with the Oracle Application Server. See the Oracle Express Server Web Products Installation Guide for more information or if you need to configure the web listener manually.

Also, be sure that Oracle Express Server is running before you install the OFA product because OFA attaches to the Express Server. If the server is not running, the installation will not succeed.

> **Note:** If you are installing on UNIX, see the section entitled "Setting Operating System Privileges in UNIX" in this chapter for important information on setting privileges.
>
> You need to set these privileges after you install the Express Server but before you install OFA and Budgeting & Planning.

In conjunction with OFA, prepare the OFA Super Administrator database. To do so, you must input database information and (if your organization needs more than the seven dimensions provided in Budgeting & Planning) create user-defined dimensions.

Input the following information to prepare the Super Administrator database:

- The language used by your end users
- The month in which your current fiscal year begins
- The current fiscal year and month in your FDM RDBMS (if applicable)
- The fiscal years for which your organization has actual and forecasted data
- The username and the password of the FDM Schema Owner
- The TNSNAMES alias for the FDM database instance

After installing OFA the next task is setting up a Super Administrator workstation. Refer to the OFA documentation for the steps you need to follow to complete this task. After creating the Super Administrator workstation, you can prepare the Super Administrator database.

Perform the following steps to complete this task:

1. Launch the OFA client interface and log in as the Super Administrator.
2. Select the default language for the application.
3. Select Maintain>Time from the menu bar to add the number of years for which you have actual and forecasted data.

   You are prompted to type the fiscal start month of your fiscal year before you can add the years for actual and forecasted data. Add years in chronological order, starting with the oldest historical year.

If you are not adding a user-defined dimension, you can exit OFA now. This completes the required Super Administrator database preparation.

This release of Budgeting & Planning enables you to add one user-defined dimension to the Budgeting & Planning data model. If you are adding a dimension, then proceed to the next step. Be sure to refer to the section entitled "Adding a User-defined Dimension" in this chapter for additional steps to follow later to add a dimension to the Budgeting & Planning data model.

Note: You must create at least one dimension value for your user-defined dimension.

---

4. Create your dimension from Maintain>Dimension on the menu bar.
5. Create at least one of your dimension values from Maintain>Dimension Values on the menu bar.
6. Exit OFA.

Completing the Super Administrator's Database Set Up

The next two steps are done through Express Administrator.

First, test the OCI SQL connection to the FDM RDMBS at this point by following the next step.

7. Issue these Express commands in Express Administrator:

```
sql.dbms = 'ORACLE'
sql connect '<user>/<password>@<dbname>'
sql disconnect
```

where user and password are the username and the password of the FDM Schema Owner, and dbname is the TNSNAMES alias (in the tnsnames.ora file installed on the same platform as the Express / OFA server) for the FDM database instance.

These commands should execute with no errors. If you receive an error, refer to the *Oracle Express Server Installation and Configuration Guide* for your platform, and to the applicable SQL*Net or Oracle Net8 documentation.

Caution: Resolve any connection problems before proceeding with the rest of the installation. If you do not, you will not be able to successfully complete the installation.

---

8. Back up the following OFA database files from the OFA Super Administrator's directories, by copying them to another name or a different directory:

   From the .../super/users directory, back up the super.db database file.

   From the.../super/shared directory, back up the ofas.db.

Preparing the Super Administrator database is now complete.

Installing the OFSA Group of Applications

If your organization intends to use other OFSA applications with Budgeting & Planning, the next step is to install the FDM database and the OFSA group of applications. Refer to the appropriate chapters in this guide for steps to follow to complete this phase of this installation.

On UNIX, the OFSA server software package must be installed on your server, even if you are not planning to use any other OFSA applications. If you are not using any other OFSA applications, you do not need to install the FDM database.

On an NT server, run Oracle Installer on the OFSA client-side CD-ROM and install Oracle Budgeting & Planning Server 4.5 from the Oracle Financial Services Applications Servers 4.5 entry.

Budgeting & Planning Server Installation

After the server side of the OFSA installation is complete (including the Budgeting & Planning application) the next step is to install and attach two additional

databases that are specifically designed for Budgeting & Planning. These databases are called FSBPTOOL and FSLANG.

The FSBPTOOL database contains all Budgeting & Planning Express-based programs, as well as some working dimensions and temporary working variables.

The FSLANG database contains descriptions and labels for all objects in the Budgeting & Planning model created as OFA-enabled as well as descriptions and labels for each of the seeded dimension values for every OFA-enabled dimension.

Note: Before you install these two databases you must set up Oracle Application Server, Oracle Express Server, and the OFA Super Administrator database.

Installing the FSBPTOOL and FSLANG Databases

The Budgeting & Planning application attaches these two databases each time you start up the application. Complete the following steps to create both of these databases.

These steps include directions for installing these databases on both UNIX and NT platforms.

1. **If you are using UNIX...**

   a. Log onto the server using the Oracle account.

   b. Change directory to the OFSA_INSTALL/<ofsa release>/ofsbpsrv directory.

   c. Execute the install_eif script to copy the required Express export and setup files into the CODE directory for the OFA Super Administrator.

      This script should prompt for the path, and copy the .EIF files, the STD_REP.DAT (Standard Report File), as well as setup.txt. It is important to supply the full path explicitly. Do not use environment variables here.

   If you are using NT...

      This step is not necessary because you specified the path to the CODE directory for the OFA Super Administrator during the server installation step for Budgeting & Planning.

2. Start up Express Administrator, connecting to the Express Server where OFA is installed.

If you are using UNIX...

Log on using the UNIX Oracle account.

If your server is running more than one instance of Express Server, be sure to supply the correct UUID string.

3. Using either the File>Open menu option or the Open Database icon, navigate to the server directory containing the OFA Super Administrator database (.../super/users/super.db) and attach it read-write.

4. Again using either the File>Open menu option or the Open Database icon, navigate up one directory level and open the code directory (.../super/code).

   To verify that this is the right directory, select the All Files (*.*) filter in the Files of type: text box. You should see ofaserve.db, ofatools.db and other OFA database files as well as the ofacdcf.cfg file. You will also see the std_rep.dat and setup.txt files.

   If you are using UNIX...

   You should also see the fsbptool.eif and fslang.eif files.

   If you are using NT...

   You should see the fsbptool.db and fslang.db files.

   Caution: If either UNIX or NT files for the appropriate platform are missing, or if the std_rep.dat or setup.txt files are missing, you must find and then copy them into this directory before proceeding.

   Do not open a database in this dialog.

5. Click Cancel. On most servers this will set the code directory as the working directory for the remainder of this Express session.

6. Open the Express command window and the issue the `chdir` command to see the current directory setting. If the current Express working directory is not set to the code directory referred to in the previous step, issue the `chdir` command again, supplying the full path.

   As an example:

   ```
   chdir /app/ofa/super/code
   ```

Leave the command window open so you can type the commands in the following steps.

Verify that the current directory is the *code* folder by issuing the `chdir` command without arguments. This returns the current working directory. Also verify that the STD_REP.DAT file is in the current working directory.

7. Execute the following commands to create or attach the FSBPTOOL and FSLANG databases:

```
infile setup.txt
update
call fs.setup
```

The FS.SETUP program can run for a few minutes. When the program finishes running, the database list that it reports should include these databases, attached in this order, followed by the standard Express databases:

```
SUPER     R/W  UNCHANGED ...
FSBPTOOL  R/W* UNCHANGED ...
FSLANG    R/O  UNCHANGED ...
```

* On NT servers, FSBPTOOL is attached R/O.

Once you have completed these steps and installed the FSBPTOOL and FSLANG databases the next task is to create the Budgeting & Planning structures and data in the Super Administrator database.

Recovery Procedures

If you encounter any errors in steps 8, 9 or 10, contact Oracle Support Services to resolve the problem. These steps involve programs that you need to run and errors indicate problems that you need to correct before continuing with the installation.

After resolving your errors you must reset your system so that you can re-run the programs in steps 8, 9 and 10. To do this, exit from Express Administrator and restore the database files that you backed up in step 8 of the "Before Installing Budgeting & Planning". Repeat steps 2 through 6 of the previous section, then make sure that the super, FSBPTOOL and FSLANG databases are attached as indicated in step 7. When you are finished with step 6, complete steps 8 through 10.

Creating the Budgeting & Planning Structures and Data

The numbering for the following steps continues the preceding sequence. This means that the steps in the following sections need to be performed in the same, uninterrupted Express Administrator session.

> **Caution:** If you must leave Express Administrator and complete the following steps at a later time, use a new Express Administrator session. Make sure that the Super Administrator database and the FSBPTOOL and FSLANG databases are attached exactly as shown in step 7.

Before proceeding to this next step, make sure that the required databases are attached in the correct order and the correct mode (refer to step 7). FS.ADDTOSUPER initializes the OFA database environment, attaching the following three OFA databases in read-only mode: OFASERVE, OFATOOLS, and OFALANG.

To complete this task, perform the following steps:

8. Run FS.ADDTOSUPER using the following Express command:

```
call fs.addtosuper
```

This program can run for several minutes.

If you are using UNIX...

The FS.ADDTOSUPER program calls FS.ALLCOMPILE to compile programs in the FSBPTOOL database. The output from this step can be found in a file named allcompile.log in the current server directory after FS.ADDTOSUPER is finished.

Coordinating Budgeting & Planning Metadata with the FDM Database

This task coordinates Budgeting & Planning metadata in the Super Administrator database with the FDM database.

To complete this task, perform the following steps:

9. Run FS.SET_META with the OCI connect string as the only argument.

```
call fs.set_meta ('<user>/<password>@<dbname>')
```

where user and password are the username and password of the FDM Schema Owner, and dbname is the TNSNAMES alias for the FDM database instance.

This task sets up a mapping between certain OFSA leaf columns and their corresponding dimensions in the Budgeting & Planning data model.

Note: The password is not stored anywhere in OFA or in Budgeting & Planning.

If you created a new dimension as part of the preparations for the OFA Super Administrator database then you need to complete the following step. If you did not create a new dimension, then proceed to the section entitled Setting Operating System Privileges in UNIX.

Adding a User-defined Dimension

If you created a user-defined dimension as part of your preparations for the OFA Super Administrator database you need to complete the following step to make sure it is incorporated into the Budgeting & Planning data model.

Make sure that the required databases are attached exactly as presented in step 7. For this step, the OFA database environment established in step 8 is also required. If the OFASERVE and OFATOOLS databases are not attached, then call the OFAENV.STARTUP program to initialize this OFA database environment before preceding with step 10.

10. Call FS.USER_DEF_DIMS, passing the Express name of your user-defined dimension as the argument. The following command, shown as an example, uses a user-defined dimension called UDF1.

```
call fs.user_def_dims ('UDF1')
```

Issue an UPDATE command to save your changes.

Setting Operating System Privileges in UNIX

On UNIX servers, it may be necessary to set the file access control list (facl) privileges for FSBPTOOL and FSLANG databases so that all users can attach to them. All Budgeting & Planning users should have read-only access to these databases.

For further information on this subject refer to the chapter entitled "Controlling Access" in the *Oracle Express Server Installation and Configuration Guide for UNIX, Release 6.3*, and the chapter entitled "Installing the Software" in the *Oracle Financial Analyzer Installation and Upgrade Guide, Release 6.3*.

The next step ensures that both the FSBPTOOL and FSLANG databases will automatically attach when the OFA environment is started.

Defining FSBPTOOL as the Primary Custom Database

It is important that both the FSBPTOOL and FSLANG databases attach when OFA is launched. If these databases do not attach, administrators cannot send Budgeting & Planning distributions and workstations cannot receive them. These databases also need to be attached while executing the Budgeting & Planning client or when running the Budgeting & Planning data movement routines.

To define FSBPTOOL as the primary custom database, complete this step:

1. In the .../code directory for the Super Administrator edit the ofacdcf.cfg configuration files as follows:

   ```
   OFALCNAME=fsbptool
   ```

   Editing the configuration files as shown causes the FSBPTOOL and FSLANG databases to automatically attach whenever the OFA environment is started. If you are installing on UNIX, use lower case.

   Note: The FSBPTOOL database should always be attached in read-only mode from this point forward. Any Express-based Budgeting & Planning program, such as for data movement, that attempts to initialize the OFA database environment will generate an Express error if FSBPTOOL is attached read-write.

   FSBPTOOL is defined as the primary OFA custom database. It contains a program called LC.STARTUP.PRG, which OFA executes automatically after it attaches FSBPTOOL. This program attaches the FSLANG database, which is defined as a secondary custom database. You can modify this program to attach any other user-defined databases that you want to define as a secondary OFA custom databases.

Backing Up the Budgeting & Planning and OFA Databases

Now is a good time to make backup copies of the following database files from the OFA super administrator's subdirectories.

Back up the following files:

From the .../super/code directory, back up the fsbptool.db and fslang.db database files.

From the .../super/users directory, back up the super.db database file.

From the.../super/share directory, back up the ofas.db file.

By saving these database files, you can restore your system to the post-installation state without having to repeat the server installation process. Save these with a different name than the prior backups.

Configuring the OFA Subordinate Administrators

Each OFA subordinate administrator owns its own set of OFA subdirectories on the OFA server. For each subordinate administrator that you have already defined or that you will define in the future for use with Budgeting & Planning, you must complete the following steps:

1. Copy fsbptool.db and fslang.db database files from the Super Administrator's Code directory into the Code directory for the subordinate administrator.
2. Edit the subordinate administrator's ofacdcf.cfg file as indicated in the Defining FSBPTOOL as the Primary Custom Database section.

This ensures that the FSBPTOOL and FSLANG databases automatically attach to the subordinate administrator and all of the thin client workstations served by the subordinate administrator every time the OFA environment is active.

Administering the Budgeting & Planning Databases

Now that the Budgeting & Planning database objects have been created in the OFA Super Administrator database, you need to develop the tiered hierarchy of OFA workstations and setup the client-side Budgeting & Planning application for end user access over the web. Refer to the Configuring the Web Listener and Java Client section of this chapter for detailed information on this installation process.

Then proceed with the steps outlined in "Administering the OFSA and Budgeting & Planning Databases" chapter in the *Oracle Budgeting & Planning Reference Guide Release 4.5*. The steps in this chapter include executing data movement routines to

import data from the OFSA relational database and distributing the Budgeting & Planning structures to the super shared database and down the tiered OFA user hierarchy to your end users.

Configuring the Web Listener and Java Client

With this release of Budgeting & Planning end users will log in to the application by pointing their browser to a starting web page. Budgeting & Planning now requires that Oracle Application Server and its web listener be configured to handle end users log in requests. The following section will describe the necessary steps to configure the web listener.

Testing the Technology Stack

Once you have installed Oracle Application Server, Oracle Express Server, and Oracle Financial Analyzer Server, verify that the sample Oracle Web Agent applications work. See the Oracle Web Products Installation Guide for more information.

Note: If you can not view the sample Oracle Web Agent applications, you will not be able to log in using BP. Resolve any issues with the Oracle Web Agent web client before proceeding with the Budgeting & Planning installation process. See the "Troubleshooting Tips for Express Web Agent" in Chapter 1, Installation of the Oracle Express Web Products Installation Guide.

Creating the Virtual Directories

Budgeting & Planning requires one virtual directory to hold the Budgeting & Planning html start page, Budgeting & Planning, and the J-initiator executables. The process for creating the virtual directories for Budgeting & Planning is very similar to configuring the virtual directories for Oracle Financial Analyzer Server.

1. As part of your Oracle Financial Analyzer Server installation, you should have defined OFASTART and OFAWEB virtual directories for Oracle Financial Analyzer. If you have not completed the setup of the Oracle Financial Analyzer web client, refer to the Oracle Financial Analyzer Installation and Upgrade Guide to complete that product's web installation before attempting to set up the Budgeting & Planning web client.

2. To create the required virtual directory for Budgeting & Planning, open the Oracle Application Server Manager by typing the following URL into your browser:

   ```
   http://<server name>:8888/
   ```

   Replace <server name> with the name of your server. The port number 8888 is a default value. You might have chosen something else during the Oracle Application Server installation. You will be prompted to type a user name and password. This is the user name and password given during the installation of Oracle Application Server.

3. The file system directory can be anywhere on the server you select. The name of the virtual directory can also be anything you select. However, the template for the html start page is populated with OBPWEB as the default virtual directory. Changing the name of the virtual directory in which you plan store the Budgeting & Planning html start page, and Budgeting & Planning, and J-initiator executables to anything other than OBPWEB requires that you make more extensive changes to the html start page template.

4. Once the Oracle Application Server manager has opened, expand (drill down) on the tree control labeled: HTTP Listeners. You will see a list of listeners that have been created for the server. Expand (drill down) on the tree control for the listener that you configured to run Oracle Financial Analyzer. Typically the WWW listener is used. Now select the Directory page. You should see a list of actual directory paths mapping to virtual directories.

   Verify that the following directories appear in the right column:

   ```
   /OFASTART/
   /OFAWEB/
   /oew-install/
   /owp/
   ```

   If any of these directories do not appear, Budgeting & Planning will not run. Verify that you are looking at the directory page for the particular listener that Oracle Express Server, Oracle Web Agent, and Oracle Financial Analyzer were configured to use. If not, open that listener in the Oracle Application Server Manager.

5. Type the value for the file system directory in which you want to install the Budgeting & Planning start page, and Budgeting & Planning, and J-initiator executables in the File-System column on the left side. On the same row, in the Virtual Directory column on the right side, type:

```
/OBPWEB/
```

Accept the default value for the flag: NR. Hit the Apply button at the end of the page. Oracle Application Manager will now validate the existence of the file system directory you typed. If you receive an error message verify that the folder exists and that you have not made any errors.

6. You will now have to stop and start the listener in order for your directory change to take effect.

Note: For questions regarding the Oracle Express Server, Oracle Web Agent, and Oracle Financial Analyzer web installations, refer to their respective installation and configuration or upgrade guides.

Configuring the Timeout Parameters

There are two server time out parameters that could impact an end user. If the user interface is idle for any amount of time that is greater than either one of the time out parameters, the end user will see an error message and, if they have not previously saved their work, it will be lost.

The first parameter is set in the Oracle Application Server Manager. Refer to step 2 in Creating the Virtual Directories section of this chapter. Expand the HTTP Listeners node, then expand the listener where you configured Express, OFA, and Budgeting & Planning (the default name of the listener is WWW). Now, select the server page.

You will see the parameter: “CGI Timeout”. The parameter is measured in seconds, that is, 600 is equal to 10 minutes. Therefore if you set the Web Listener CGI Timeout parameter to 600, and the user leaves the application idle for 11 minutes, the CGI connection will close and the user could possibly lose data. For more information on setting the time out parameter, see the Oracle Application Server reference guide and the online help.

The second parameter is set in the Express Instance Manager. Open the Express instance manager, expand the particular Express Server instance you are using, then expand the parameters node. Now, select the WebAgent page.

You will see the parameter: Time-out. The parameter is also measured in seconds, that is, 600 is equal to 10 minutes. Therefore if you set the WebAgent Timeout parameter to 600, and the user leaves the application idle for 11 minutes, the Web

Agent connection will close and the user could possibly lose data. For more information on setting the time out parameter, see the Oracle Express Server reference guide and the online help.

Caution: End users may lose data if they fail to save their work before the Web Listener and/or the web agent connections time out. To prevent data loss, educate your end users about your time out parameter settings.

Installing the files into the virtual directory

1. Installing the Files

   If you are installing on NT...

   a. Run the Oracle installer.

   b. Under the Oracle Financial Services Web Server Applications 4.5 program group, select the Oracle Budgeting & Planning 4.5 option.

   If you are installing on Sun UNIX...

   a. See Chapter 6, "UNIX Server Installation and Configuration"

   b. When prompted to make a choice regarding what to install, select *B* for Budgeting & Planning Web Client or *F* for Full Package.

   If you are installing on HP or IBM-AIX UNIX...

   a. See Chapter 6, "UNIX Server Installation and Configuration".

   b. After the server centric software install, change directory to the OFSA_INSTALL/<ofsa release>/bpweb directory.

   c. Execute the install_bpjar script to copy the required HTML and JAR files into the virtual directory. This script should prompt for a path.

   d. When prompted for a path, type the file system directory given for the virtual directory created earlier. The script will copy the Budgeting & Planning HTML template, and Budgeting & Planning and J-Initiator executables to the directory given.

2. When prompted for a path, type the file system directory given for the virtual directory created earlier. The Oracle Installer will copy the Budgeting &

Planning HTML template, and Budgeting & Planning and J-initiator executables to the directory given.

Editing the HTML Start Page

Budgeting & Planning is designed to work with either an Internet Explorer or Netscape browser. Parameters for each browser need to be defined for your organization's environment.

If you plan to support only one browser then the code for that browser should be edited. You can disregard the code pertaining to the other browser.

If you plan to support both browsers, however, you need to edit the code for both. Parameter names are slightly different between the two but the portions that need to be edited will contain the same information.

Following the installation of the files for Budgeting & Planning you need to open and edit the following two HTML files:

- obp.html
- obpjinit.html

Editing HTML Files for Internet Explorer

Open the obp.html file and look for the HTML text between the following points:

Beginning point: <p><object

Ending point: After the path for the Oracle Financial Analyzer shared database (parameter name: dbdir)

Editing the obp.html File

The following parameters should be edited:

Internet Explorer	Edits	Purpose/Description
classid=	No editing required	
WIDTH / HEIGHT	No editing required	Defines the dimensions of the download and installation applet for Jinitiator

Internet Explorer	Edits	Purpose/Description
codebase	Type the complete URL to point to the /obpweb/obpjinit.html file.	The location */obpweb/obpjinit.html* is already contained in the codebase parameter. You need to add the preceding URL text, naming the server and any additional configuration information, such as port. This URL points to the obpjinit.html file, which contains the download and installation instructions for the version of Jinitiator and the link to the Jinitiator executable that actually runs the installation process. This is necessary for loading the correct version of Jinitiator on the end user's computer.
<param NAME= "CODE" VALUE	No editing required	
<param NAME= "ARCHIVE" VALUE	Edit only if the location of the obp.jar file is different from the default location	The default location for the obp.jar file is /obpweb/obp.jar. If you install this file in a different location, then you need to edit this parameter.
<param NAME= "type" VALUE	No editing required	
<param NAME= "service" VALUE	Edit only if the service name for Express Server is different than the default name	The default name for the Express service is ExpSrv630. If you are running multiple instances of Express Server on your system and need to specify a different service name, then that is the name to type in this parameter.
<param NAME= "oowadir" VALUE	Type the location of the Oracle Applications Server bin directory	Web-based applications use Oracle Applications Server. You need to specify the path/location for this server's bin directory.
<param NAME= " helpdir" VALUE	Type the location of the Help files	You need to specify the path (URL) to the help files for the application. The help files are found in the doc sub folder of the virtual directory where the Budgeting & Planning JAR file is installed. As an example: obpweb/doc/
<param NAME= "oowaexe" VALUE	For UNIX O/S: oowaro For NT O/S: oowa.exe	The application runs on either UNIX or NT operating systems. Type the value appropriate to the operating system you are using.

Internet Explorer	Edits	Purpose/Description
<param NAME= "port" VALUE	Type the Web Listener port	This parameter defines how the Web Listener is configured.
<param NAME= "host" VALUE	Type the name of the machine that Express Server is running on	This is the name of the hardware where Express Server has been installed.
<param NAME= "user" VALUE	Optional parameter	Type a value if you want the username in the login dialog box to default to a specific name. If you select this option, each application end user will require his or her own HTML page.
<param NAME= "dbdir" VALUE	Type the fully qualified path to the directory containing the shared database	This is the directory location of the Oracle Financial Analyzer shared database. This directory is *not* a virtual directory as used by the Web Listener.

Editing the obpjinit File

This file is never displayed by Internet Explorer.

Editing HTML Files for Netscape

Open the obp.html file and look for the HTML text between the following points:

Beginning point: <embed type=...(immediately following <COMMENT>)

Ending point: </embed> (immediately preceding </COMMENT>

Editing the obp.html File

The following parameters should be edited:

Netscape	Edits	Purpose/Description
type=	No editing required	
java_CODE=	No editing required	
java_ARCHIVE=	Edit only if the location of the obp.jar file is different from the default location	The default location for the obp.jar file is /obpweb/obp.jar. If you install this file in a different location, then you need to edit this parameter.

Netscape	Edits	Purpose/Description
WIDTH / HEIGHT	No editing required	Defines the dimensions of the download and installation applet for Jinitiator
service=	Edit only if the service name for Express Server is different than the default name	The default name for the Express service is ExpSrv630. If you are running multiple instances of Express Server on your system and need to specify a different service name, then that is the name to type in this parameter.
oowadir=	Type the location of the Oracle Applications Server bin directory	Web-based applications use Oracle Applications Server. You need to specify the path / location for this server's bin directory.
helpdir=	Type the location of the Help files	You need to specify the path (URL) to the help files for the application. The help files are found in the doc sub folder of the virtual directory where the Budgeting & Planning JAR file is installed. As an example: /obpweb/doc
oowaexe=	For UNIX O/S: oowaro For NT O/S: oowa.exe	The application runs on either UNIX or NT operating systems. Type the value appropriate to the operating system you are using.
port=	Type the Web Listener port	This parameter defines how the Web Listener is configured.
host=	Type the name of the machine that Express Server is running on	This is the name of the hardware where Express Server has been installed.
user=	Optional parameter	Type a value if you want the username in the login dialog box to default to a specific name. If you select this option, each application end user will require their own HTML page.
dbdir=	Type the fully qualified path to the directory containing the shared database	This is the directory location of the Oracle Financial Analyzer shared database. This directory is *not* a virtual directory as used by the Web Listener.

Netscape	Edits	Purpose/Description
pluginspage=	Type the complete URL to point to the /obpweb/obpjinit.html file.	The location */obpweb/obpjinit.html* is already contained in the codebase parameter. You need to add the preceding URL text, naming the server and any additional configuration information, such as port. This URL points to the obpjinit.html file, which contains the download and installation instructions for the version of Jinitiator and the link to the Jinitiator executable that actually runs the installation process. This is necessary for loading the correct version of Jinitiator on the end user's computer.

Editing the obpjinit File

Open the obpjinit.html file and look for the following HTML text:

```
<a href="http://server_name/obpweb/jinit11729.exe#Version=1.1.7.29">
```

Change server_name to the name of the machine where the jinit11729.exe is installed.

9

Budgeting & Planning Database Upgrade Process

This chapter discusses the procedure for upgrading the Oracle Budgeting & Planning Express database. This procedure supports upgrading from Budgeting & Planning Release 4.0 Express databases.

The following topics are covered in this chapter:

- Installing the Technology Stack
- Installing the Budgeting & Planning Code Databases and Files
- Upgrading the Super Administrator's Personal Database
- Completing the Database Upgrade Process

Installing the Technology Stack

Note: This chapter is intended to supplement, not replace, the installation and configuration or upgrade guides for supporting products. Refer to those documents for important information not covered here.

Before running the Budgeting & Planning Database Upgrade Process (DUP), install the following components in the stated order:

1. Upgrade the Technology Stack
    a. Install Oracle Application server, version 4.0.8.1.

b. Upgrade Oracle Express Server to version 6.3.0.1 (with patch "OWA630_P1.exe")

c. Upgrade Oracle Financial Analyzer Server to version 6.3.0.0 (with patch "6325_2.exe").

Note: When you install Oracle Application Server before upgrade/re-installing Oracle Express Server, the Oracle Express Server installation automatically configures the web listener included with Oracle Application Server. See the Oracle Express Server Web Products Installation Guide for more information, or if you need to configure the web listener manually.

---

2. Once Oracle Express Server and Oracle Application Server are installed, complete the entire Oracle Financial Analyzer upgrade process. For more information see the Oracle Financial Analyzer Installation and Upgrade Guide.

3. Upgrade the Super Administrator's personal database and the Super Administrator's shared database by installing Financial Analyzer Server version 6.3.0.0 into the directory where the super databases are located. The installation for Oracle Financial Analyzer Server version 6.3.0.0 prompts the user for the path to the super databases. The path that the install routine is looking for is the parent directory for the *users* and the *shared* directories. For example, if the super database is located in the C:\OFAPROD1\SUPER\USERS directory, the path the install routine looks for "C:\OFAPROD1\SUPER".

   When the Oracle Universal Installer installs Financial Analyzer Server, it overwrites the Oracle Financial Analyzer code databases and regenerates all the configuration files. It does not overwrite the super or the shared databases.

   If there are sub-administrators, they also need the new version of Oracle Financial Analyzer Server to be installed.

   Note: If the server is NT, use \ (back slashes) when you are entering the path in the Oracle Universal Installer rather than using the Oracle Express Server / (forward slash) convention. Also, omit any trailing slash.

Testing the technology stack

Follow the steps in the Oracle Financial Analyzer Installation and Upgrade Guide to configure Oracle Financial Analyzer web client. Log into the Oracle Financial

Analyzer web client. If you are unable to log into Oracle Financial Analyzer using the web client, contact Oracle Financial Analyzer technical support before proceeding with the Budgeting & Planning Database Upgrade Process.

> **Note:** If you cannot log in to your database using the Oracle Financial Analyzer web client, you cannot log in using Budgeting & Planning. Resolve any issues with the Oracle Financial Analyzer web client before attempting to run the Budgeting & Planning Database Upgrade Process.

Installing the Budgeting & Planning Code Databases and Files

Rename the existing FSBPTOOL and FSLANG databases in the code folder of your super administrator's Oracle Financial Analyzer installation for backup purposes and so that you do not have a name conflict when the new databases are created.

If you are using NT...

1. Run the Oracle Installer.
2. Under the Oracle Financial Services Applications Servers 4.5 program group, select the Oracle Budgeting & Planning Server 4.5 option.
3. When prompted for a path, enter the path to the super administrator. The Oracle Installer then copies the two code databases FSBPTOOL and FSLANG, the setup.txt file, and the std_rep.dat (standard reports) file into the code folder.

   The path that the install routine is looking for is the parent directory for the *users* and the *shared* directories. For example, if the super database is located in the C:\OFAPROD1\SUPER\USERS directory, the path the install routine looks for C:\OFAPROD1\SUPER.

If you are using UNIX...

1. Log onto the server using the Oracle account.
2. Change directory to the OFSA_INSTALL/<ofsa release>/ofsbpsrv directory.
3. Execute the install_eif script to copy the required Express export and setup files into the CODE directory for the OFA Super Administrator.

This script should prompt for the path, and copy the .EIF files, the STD_ REP.DAT Standard Report File), as well as setup.txt. It is important to supply the full path explicitly. Do not use environment variables here.

Upgrading the Super Administrator's Personal Database

1. Open Express Administrator. From the file menu select the open option, and attach the super's personal database read / write.If it exists in the super's personal database, delete the program FS.SETUP. From the Express Command window, verify that the super's personal database is attached first, using the DATABASE LIST function. If it is not, then issue DATABASE ATTACH SUPER FIRST where super is the name of the super's personal database.
2. Now use the CHDIR and CHDRIVE commands to change the Express Working directory to be the code folder associated the super's personal and shared databases. Reissue the CHDIR command with no arguments to verify that the current directory is actually the Super Administrator's code directory.
3. If the directory is not the directory you would expect, use the CHDRIVE command to change the current drive to the location of the Super Administrator's CODE directory. Reissue the CHDIR command with no arguments to verify that the current directory is actually the Super Administrator's CODE directory.
4. Executing the Budgeting & Planning Database Upgrade Process

   If you are using NT…

   a. Issue the following Express Commands:

   b. In order for OFAENV.STARTUP to successfully run, FSBPTOOL must be attached read / only. From the Express Command window, attach FSBPTOOL read / only and run OFAENV.STARTUP by issuing the following Express commands:

   ```
   DATABASE ATTACH FSBPTOOL RO
   CALL OFAENV.STARTUP
   DATABASE LIST
   ```

   c. Verify that super's personal database is attached read / write first, that FSBPTOOL and FSLANG are attached read / only, and that the Oracle Financial Analyzer code databases are attached read / only as well. If FSLANG is not attached, verify the OFACDCF.CFG file has fsbptool listed as the primary custom database. If it does not, update the OFACDCF.CFG file and re run OFAENV.STARTUP.

d. Now call the upgrade program

```
CALL FS.UPGRADE
```

If you are using UNIX...

a. Issue the following Express Commands:

```
INFILE SETUP.TXT
CALL FS.SETUP
UPDATE
DATABASE DETACH FSBPTOOL
DATABASE DETACH FSLANG.
```

The INFILE command creates (or recreates) the FS.SETUP program in the super database. Running the FS.SETUP program creates the new FSBPTOOL and FSLANG version 4.5 databases. The UPDATE command writes the new databases to the disk, and the detaching commands set the environment for the rest of the Budgeting & Planning Database Upgrade Process

b. In order for OFAENV.STARTUP to successfully run, FSBPTOOL but be attached read / only. From the Express Command window, reattach FSBPTOOL read / only and run OFAENV.STARTUP, by issuing the following Express commands:

```
DATABASE ATTACH FSBPTOOL RO
CALL OFAENV.STARTUP
DATABASE LIST
```

c. Verify that the super's personal database is attached read / write first, that FSBPTOOL and FSLANG are attached read / only and that the Oracle Financial Analyzer code databases are attached read / only as well. If FSLANG is not attached, verify the OFACDCF.CFG file has fsbptool listed as the primary custom database. If it does not, update the OFACDCF.CFG file and re run OFAENV.STARTUP.

If you have installed OFA on UNIX, be sure to use lower case.

d. In order for FS.UPGRADE to successfully run, FSBPTOOL must be attached read / write. From the Express command window, detach FSBPTOOL and reattach it read / write before running FS.UPGRADE, by issuing the following Express commands:

```
DATABASE DETACH FSBPTOOL
DATABASE ATTACH FSBPTOOL RW
CALL FS.UPGRADE
```

Your super personal database is now upgraded to version 4.5 of Budgeting & Planning. You should see a message that informs you that the upgrade was successful.

Completing the Database Upgrade Process

Once the super administrator's personal database has been upgraded, you need to upgrade the super administrator's shared database and any sub administrator's personal and shared databases.

Note: In this release of Budgeting & Planning, users have Read Only access to the shared database. Existing Budget users should submit any data to the shared database. Create any new users as External Users. See the *Oracle Budgeting & Planning Reference Guide* for more information about Oracle Financial Analyzer users.

---

1. Log into the super administrator's personal database using the Oracle Financial Analyzer client. From the manage menu, first select distribution, then the distribute structure option. In the distribute structure interface select the Auto DUP item. Using the Add action, distribute the dimension and its seeded dimension value to all sub administrator users.

Caution: Do not attempt to distribute any other objects with the Auto DUP item.

---

2. Submit the task to the task processor and then launch the task processor if it is not already running. Once the task has been processed, log into each of the subordinate administrator's personal databases to process the distribution. If you have additional levels of subordinate administrators, repeat the distribution of the Auto DUP item to all lower levels of subordinate administrators.

3. Once the Auto DUP process has run distribute the Non-OFA Budgeting & Planning Database Objects and the One-way Data Custom Distribution objects to the shared database and any subordinate administrators.

10

FDM Database Installation

This chapter provides information on installing the Oracle Financial Data Manager (FDM) database. The specific topics covered in this chapter include:

- Installing the Oracle Applications
- Setting up the Physical Structure of the Oracle Database
- Configuring the FDM Database
- Creating the FDM Database

You can find additional information on installing and configuring the Oracle database for the Oracle Financial Services Applications (OFSA) group of applications in the reference and installation guides associated with the Oracle database and applications you are installing.

Installing the Oracle Applications

This section addresses Oracle application installation issues as they pertain to the OFSA group of applications. Refer, also, to the platform-specific installation guide provided by Oracle for further, detailed installation instructions.

Installing the Oracle Database-related Components

Following is a list of Oracle-related components required for a minimum installation. You can find these components on the media distributed with the Oracle RDBMS software.

- Oracle Installer
- Oracle8i Server‡
- PL/SQL‡
- Oracle NET8‡
- SQL*Plus‡
- Oracle Server Manager: Line Mode

> **Note:** For components marked with the following notation (‡), you need the appropriate version that is shipping with Oracle 8.1.6.x

The following is needed if you are upgrading from Oracle7

- Migration Utility

Checking the Installation for Errors or Failures

Check the installation log files for any errors or failures that might have occurred during the installation process. If more detailed information is necessary, refer to the reference guides associated with the Oracle applications you are installing.

Setting up the Physical Structure of the Oracle Database

The physical structure of the Oracle database includes the directory structure, parameter files, datafiles, control files and redo logs. Each of these components are described in this section.

Structure Files

The physical structure of the Oracle database is contained in the three file categories listed and described in the following table.

File Type	Contents
Datafiles	The datafiles are physical files containing all of the database data. These files include the physical database structures such as tables and indexes. Your database can have one or more datafiles.
Redo Log Files	The redo log files contain the redo log, which records and retains all changes made to the data. Should a system failure prevent modified data from being written to the datafiles, the changes can be obtained from the redo log so that work is not lost. Your database can have one or more redo log files.
Control Files	Control files record the physical structure of the database, such as the database name, the names and locations of the datafiles and redo log files and the time stamp of the database creation. Your database should have at least two control files. However, three is recommended.

Parameter Files

The initialization parameter file is a text file that containing a list of parameters and a value for each parameter. The file should be written in your default character set. Specify values in the parameter file that reflect your installation.

Parameter files contain the configuration parameters for the Oracle database. Setting up parameter files are a prerequisite to starting the Oracle instance.

There are two Oracle parameter files. In the following list, <dbname> is the database name you have selected and need to replace. The dbname is also referred to as the SID.

```
init<dbname>.ora
config<dbname>.ora
```

Oracle parameter files contain information and instructions such as those in the following list:

- Database name
- Memory capacity assigned to the System Global Area (SGA)
- Instructions for handling filled redo log files
- Name and location of the control file(s)
- Names of private rollback segments in the database
- Parameters that name things (such as files)
- Parameters that set limits (such as maximums)
- Parameters that affect capacity (called variable parameters)

  An example of a variable parameter is a DB_BLOCK_BUFFERS parameter, which specifies the number of data blocks allocated for the SGA in the computer's memory.

Required Parameters for OFSA

The FDM database requires the following initialization parameters:

- **compatible**

  FDM requires that this parameter is set to 8.1.6 or higher.

  COMPATIBLE lets you use a new release, while at the same time guaranteeing backward compatibility with an earlier release. This ability is helpful in case it becomes necessary to revert to the earlier release.

- **dml_locks**

  FDM requires that this parameter is set to at least 200.

  A DML lock is a lock obtained on a table that is undergoing a DML operation (insert, update, delete). DML_LOCKS specifies the maximum number of DML locks--one for each table modified in a transaction. The value should equal the grand total of locks on tables currently referenced by all users. For example, if three users are modifying data in one table, then three entries would be required. If three users are modifying data in two tables, then six entries would be required.

- **job_queue_processes**

  FDM requires that this parameter is set to at least 1 (maximum value is 36).

JOB_QUEUE_PROCESSES specifies the number of SNPn job queue processes per instance (SNP0, ... SNP9, SNPA, ... SNPZ). Job queue processes process requests created by DBMS_JOB.

- **max_enabled_roles**

  FDM requires that the max_enabled_roles parameter is set to at least 60. This is because the FDM database creation and database upgrade processes create a number of seeded roles in the database instance.

  MAX_ENABLED_ROLES specifies the maximum number of database roles that users can enable, including roles contained within other roles.

- **open_cursors**

  Specifies the maximum number of open cursors (context areas) a session can have at once. This constrains a session from opening an excessive number of cursors. The FDM Administration application requires 100 as a minimum, however Oracle recommends a value of 200 to accommodate OFS applications multiprocessing.

Performance Parameters for OFSA

You can use initialization parameters to do the following:

- Optimize performance by adjusting memory structures.

  Example: The number of database buffers in memory

- Set some database-wide defaults.

  Example: How much space is initially allocated for a context area when it is created

- Set database limits.

  Example: The maximum number of database users

- Specify names of directories.

  Examples: BACKGROUND_DUMP_DEST, USER_DUMP_DEST and CORE_DUMP_DEST

Parameters that should be evaluated to enhance the performance of the OFSA database include the ones that follow. Refer to the *Oracle8i Reference* for more information on these parameters.

- **shared_pool_size**

  Specifies the size of the shared pool in bytes. The shared pool contains shared cursors and stored procedures. Larger values improve performance in multi-user systems.

- **shared_pool_reserved_size**

  This is shared pool space that is reserved for large, contiguous requests for shared pool memory. This parameter, along with the SHARED_POOL_RESERVED_MIN_ALLOC parameter, can be used to avoid performance degradation in the shared pool in situations where pool fragmentation forces Oracle to search for any free chunks of unused pool to satisfy the current request.

- **shared_pool_reserved_min_alloc**

  Controls the allocation of reserved memory. Requests for memory allocations greater than the value of the parameter allocate space from the reserved list if the memory chunk was not found on the shared pool free list.

- **db_block_buffers**

  This is the data cache. The larger the cache, the more data Oracle can hold in memory. A large data cache is desirable because it reduces I/O calls to the operating system. Increasing the data cache keeps queried data in memory longer, which reduces the I/O load.

  You can determine the total buffer cache available by using the following formula: DB_BLOCK_BUFFERS * DB_BLOCK_SIZE.

- **cursor_space_for_time = true**

  When set to true, this parameter causes shared SQL areas to be pinned in the shared pool. The shared SQL area is not aged-out when there is an open cursor that references the shared SQL. This parameter also causes the private SQL area allocated to each cursor, to be maintained in the library cache. Since Oracle must scan the library cache to determine if the SQL statement is present when set to true, then this check is not needed.

You should set this parameter to true only if the shared pool is large enough to hold all open cursors simultaneously.

- **db_block_size**

  Specifies the size, in bytes, of Oracle database blocks. The typical value for OFSA is 16K. The DB_BLOCK_SIZE should be a factor of the operating system block size.

- **db_file_multiblock_read_count**

  This parameter is used for multi-block I/O and specifies the maximum number of blocks read in one I/O operation during a sequential scan. The total number of I/Os needed to perform a full table scan depends on factors such as the following:

  - The size of the table
  - The multi-block read count
  - Whether parallel query is being used for the operation

- **db_block_lru_latches**

  Specifies the upper bound of the number of LRU latch sets. Set this parameter to a value equal to the desired number of LRU latch sets. Oracle decides whether to use this value or reduce it based on a number of internal checks. If the parameter is not set, Oracle calculates a value for the number of sets.

- **log_buffer**

  Specifies the amount of memory, in bytes, that is used when buffering redo entries to a redo log file. Redo log entries contain a record of the changes that have been made to the database block buffers. The LGWR process writes redo log entries from the log buffer to a redo log file.

- **log_checkpoint_interval**

  Specifies the frequency of checkpoints in terms of the number of redo log file blocks that are written between consecutive checkpoints. Regardless of this value, a checkpoint always occurs when switching from one online redo log file to another. If the value exceeds the actual redo log file size, checkpoints occur

only when switching logs. The checkpoint frequency is one of the factors that impacts the time required for the database to recover from an unexpected failure.

Caution: Frequent checkpointing can cause excessive I/O and writes to the disk.

- **sort_area_size**

  Specifies the maximum amount, in bytes, of Program Global Area (PGA) memory to use for a sort. After the sort is complete and all that remains to do is to fetch the rows, the memory is released down to the size specified by SORT_AREA_RETAINED_SIZE. After the last row is fetched, all memory is freed. The memory is released back to the PGA, not to the operating system.

  Increasing SORT_AREA_SIZE size improves the efficiency of large sorts. Multiple allocations never exist; there is only one memory area of SORT_AREA_SIZE for each user process at any time.

  If more space is required, temporary segments are created on the disk.

- **disk_asynch_io**

  This parameter can be used to control whether I/O to datafiles, control files and log files are asynchronous. If a platform supports asynchronous I/O to disk, it is recommended that this parameter be left at the default setting. However, if the asynchronous I/O implementation is not stable, this parameter can be set to FALSE to disable asynchronous I/O. If a platform does not support asynchronous I/O to disk, this parameter has no effect.

- **enqueue_resources**

  Sets the number of resources that can be concurrently locked by the lock manager. The default value of ENQUEUE_RESOURCES is derived from the SESSIONS parameter and should be adequate, as long as DML_LOCKS + 20 are less than ENQUEUE_RESOURCES.

- **parallel_max_servers**

Specifies the maximum number of parallel query servers or parallel recovery processes for an instance. Oracle increases the number of query servers, as demand requires, from the number created at instance startup, up to this value. Proper setting of this parameter ensures that the number of query servers in use does not cause a memory resource shortage during periods of peak database use.

If PARALLEL_MAX_SERVERS is set too low, some queries may not have a query server available to them during query processing. Setting PARALLEL_MAX_SERVERS too high leads to memory resource shortages during peak periods, which can degrade performance.

- **parallel_min_servers**

  Specifies the minimum number of query server processes for an instance. This is also the number of query server processes Oracle creates when the instance is started.

- **optimizer_percent_parallel**

  Specifies the amount of parallelism that the optimizer uses in its cost functions. The optimizer percent parallel default of 0 means that the optimizer selects the best serial plan. A value of 100 means that the optimizer uses each object's degree of parallelism in computing the cost of a full table scan operation. Low values favor indexes, and high values favor table scans.

Packages

Pinning the large packages in the following list reduces fragmentation in the shared pool. The following is provided in a script called pin.sql and is located in OFSA_INSTALL/dbs/<ofsa release>/master. You need to edit this script for your particular database.

> **Note:** In this chapter, OFSA_INSTALL is the convention used to indicate where the OFSA group of applications is installed in your directory structure.

Run this script each time you start up your database.

REM * This script pins frequently used packages in the shared pool

```
connect internal
execute dbms_shared_pool.keep('DBMS_STANDARD');
execute dbms_shared_pool.keep('DBMS_SHARED_POOL');
execute dbms_shared_pool.keep('DBMS_OUTPUT');
execute dbms_shared_pool.keep('STANDARD');
execute dbms_shared_pool.keep('DBMS_UTILITY');
execute dbms_shared_pool.keep('<dbowner>.OFSA_SQL');
execute dbms_shared_pool.keep('<dbowner>.OFSA_DBA');
```

Objects can be manually pinned in the shared pool in cases where application code is being used over and over.

Database Tablespaces and Datafiles

A database is divided into logical storage units called tablespaces, which are used to group logical structures together. Tablespaces are made up of one or more files that are located either on UNIX file systems or on RAW devices.Tablespaces are used to:

- Control disk space allocation for database data and indexes
- Assign specific space quotas for database users
- Control availability of data by taking individual tablespaces online or offline
- Perform partial database backup or recovery operations
- Allocate data storage across devices to improve performance.

FDM Tablespaces

A tablespace consists of one or more datafiles. The FDM database creation and database upgrade processes require a minimum of two tablespaces with the naming conventions listed in the following table.

Caution: Your FDM database must have a minimum of two tablespaces, using the naming conventions in the following table

Tablespace Name	Contents and Usage
DATA_TS	Location of the OFSA tables and data necessary for Upgrade Process

Tablespace Name	Contents and Usage
INDEX_TS	Location of the OFSA indexes necessary for Upgrade Process

You also need to define a temporary tablespace and a rollback tablespace. The recommended naming convention for these tablespaces is TEMP_TS and ROLLBACK_TS, respectively.

Instrument tables such as MORTGAGES, DEPOSITS, CONSUMER_LOAN, COMMERCIAL_LOAN or LEDGER_STAT should be moved from the DATA_TS to their own tablespaces. You should use table partitioning as a method of placing segments of a table's data into different tablespaces. See the section entitled "Table and Index Partitioning" in this chapter for additional details.

FDM Datafiles

Examples of the naming conventions for datafiles associated with each of the required tablespaces are provided in the following table.

Tablespace Name	Naming Convention Example
DATA_TS	<dbname>_DATA_01.DBF
	<dbname>_DATA_02.DBF
INDEX_TS	<dbname>_INDEX_01.DBF
	<dbname>_INDEX_02.DBF

Follow the numbering convention in this example for multiple datafiles within tablespaces. The number of datafiles for a tablespace can vary from database to database.

Datafile Location

The location of the datafiles for your tablespaces varies based on many factors, including whether your database is on raw devices or file systems.

If the database is on raw devices the datafiles are defined by a logical volume management program such as Volume Manager and must be mapped into logical volumes.

If the database is on file systems, the datafiles are located in directories that can use the following naming conventions: /db/d02/oradata/<dbname>,

/db/d03/oradata/<dbname>, /db/d04/oradata/<dbname>, and so forth. The datafiles are located in the **oradata/<dbname>** subdirectory.

Table and Index Partitioning

Data partitioning is a new feature of Oracle8, designed to enable users to decompose tables into smaller and more manageable pieces called partitions. All partitions of a table or index have the same logical attributes, although their physical attributes can be different. For example, all partitions in a table share the same column and constraint definitions; and all partitions in an index share the same index columns. However, storage specifications and other physical attributes such as PCTFREE, PCTUSED, INITRANS, and MAXTRANS can vary for different partitions of the same table or index.

The table or index can be divided into partitions, based on a range of key values. Each partition can be operated on independently. For example, a table partition can be recovered, undergo DML (insert, update, delete) transactions, be analyzed and so forth without affecting the other partitions.

Each partition is stored in a separate segment. You can also store each partition in a separate tablespace, which has the following advantages:

- I/O load balancing by mapping partitions to disk drives
- Performance improvements using partition elimination
- Containing the impact of damaged data
- Independent backup and recovery of each partition

The primary benefit of the partitioning option is ease of maintenance for large tables and improved reliability, which is important for both online transaction processing (OLTP) and data warehousing environments. However, performance improvement using partition elimination is also an important benefit of partitioning.

Partition elimination occurs when the optimizer selects an execution plan that skips partitions not needed for the query. This partition elimination takes place at run time, when the execution plan references all partitions. Partition elimination provides a performance benefit by reducing the number of partitions that must be accessed to satisfy a query.

Table partitioning can be useful in OFSA by partitioning the primary transaction (instrument) tables. Examples of these tables include DEPOSITS, LEDGER_STAT COMMERCIAL_LOAN, CONSUMER_LOAN, MORTGAGES and CREDIT_CARD.

You must identify a partition key and a range of values on which to partition for each table. These key ranges represent the less-than-values to be used to partition

the data. To create a partition table or index use the new enhanced create table and create index syntax to specify the partition key(s) and range of values for each partition.

Partitioning Example

CREATE TABLE "DEPOSITS" ("IDENTITY_CODE" NUMBER(10, 0) NOT NULL ENABLE, "ID_NUMBER" NUMBER(25, 0) NOT NULL ENABLE, "GL_ ACCOUNT_ID" NUMBER(14, 0) NOT NULL ENABLE, "ORG_UNIT_ID" NUMBER(14, 0) NOT NULL ENABLE, "COMMON_COA_ID" NUMBER(14, 0) NOT NULL ENABLE, "DETAIL_RECORD" NUMBER(5, 0), "AS_OF_DATE" DATE NOT NULL ENABLE,...)

PARTITION BY RANGE (PRODUCT_ID)

(PARTITION DEPOSIT_P1 VALUES LESS THAN ('211146')

TABLESPACE DEPOSIT01,

PARTITION DEPOSIT_P2 VALUES LESS THAN ('213083')

TABLESPACE DEPOSIT02,

PARTITION DEPOSIT_P3 VALUES LESS THAN ('221042')

TABLESPACE DEPOSIT03,

PARTITION DEPOSIT_P4 VALUES LESS THAN ('222253')

TABLESPACE DEPOSIT04,

PARTITION DEPOSIT_P5 VALUES LESS THAN ('262021')

TABLESPACE DEPOSIT05,

PARTITION DEPOSIT_P6 VALUES LESS THAN ('262101')

TABLESPACE DEPOSIT06,

PARTITION DEPOSIT_P7 VALUES LESS THAN (MAXVALUE)

TABLESPACE DEPOSIT07)

PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255 NOLOGGING

STORAGE (INITIAL 25600000 NEXT 25600000 MINEXTENTS 1 MAXEXTENTS UNLIMITED PCTINCREASE 0 FREELISTS 4 FREELIST GROUPS 1 BUFFER_POOL DEFAULT);

CREATE INDEX "DEPOSITS_201" ON "DEPOSITS" ("TP_COA_ID", "ORG_UNIT_ID") local

(partition deposits_index_201_p1 tablespace deposit_index01,

partition deposits_index_201_p2 tablespace deposit_index02,

partition deposits_index_201_p3 tablespace deposit_index03,

partition deposits_index_201_p4 tablespace deposit_index04,

partition deposits_index_201_p5 tablespace deposit_index05,

partition deposits_index_201_p6 tablespace deposit_index06,

partition deposits_index_201_p7 tablespace deposit_index07)

PCTFREE 10 INITRANS 2 MAXTRANS 255 STORAGE (INITIAL 7168000 NEXT 104800 MINEXTENTS 1 MAXEXTENTS UNLIMITED PCTINCREASE 0 FREELISTS 4 FREELIST GROUPS 1 BUFFER_POOL DEFAULT) nologging;

You should identify a key range that helps balance partitions into evenly sized segments. It is also helpful if the partition key is the same key used to retrieve the data. This may not always be possible but should be considered.

There are also several restrictions to using partitions, including those listed in the following table.

Restriction	Explanation
Datatype Restrictions	At this time partitioned tables cannot have any columns with LONG or LONG RAW datatypes, LOB datatypes (BLOB, CLOB, NCLOB, or BFILE) or object types.
DATE datatype	If a table or index is partitioned on a column that has the DATE datatype and if the NLS date format does not specify the century with the year, the partition descriptions must use the TO_DATE function to specify the year completely. Otherwise, you cannot create the table or index.
Clusters cannot be partitioned	(None)
Optimizer Restrictions	Oracle8 COST BASED Optimizer supports partitions. Rule Base Optimizer is not *partition aware* and does not support partition elimination. Any application using Rule Base Optimizer does not gain any performance benefits from using partitioned tables or indexes. However, these applications can make use of ease of administration and availability features of partitioned tables.

Restriction	Explanation
Physical Restrictions	Partitioned tables cannot span multiple databases. They must be within one instance. For more information refer to the Oracle8i Server documentation and Chapter 11, "Upgrading from OFSA 3.5/4.0" in this manual.

Configuring the FDM Database

This section discusses specific configuration tasks you need to complete to create a well-organized area that is compliant for Optimal Flexible Architecture (OFA).

Creating the Working Directories

You need to create the working directories for the Oracle database. From the $ORACLE_BASE/admin directory, create the following database directory:

```
mkdir <dbname>
```

Under the new <dbname> directory, create the following subdirectories:

```
bdump
bkup
cdump
create
pfile
udump
adhoc
```

Setting init Parameters

Complete the steps in this section to modify the init parameters. A generic version of the **init.ora** file is located in the **$ORACLE_HOME/dbs** directory. You can either copy this file and edit it or create your own.

The size parameters noted in these steps are recommended settings, some of which are defined as minimum recommended settings. You may need to make further modifications to these setting based, on your preferences or the complexity of your system.

1. From the **$ORACLE_BASE/admin/<dbname>/pfile** directory, edit or create the following file:

   ```
   init<dbname>.ora
   ```

2. Add the SORT_AREA_SIZE parameter and set it to at least 5 MB.

   This parameter is not included in the default init.ora file.

3. Increase the SHARED_POOL_SIZE to at least 5 MB.

4. Add the DB_BLOCK_SIZE parameter and set it to at least 16384.

   This parameter is not included in the default init.ora file.

   If your platform does not support a 16K block size, use an 8192 block size instead.

5. Change all occurrences of the instance name (<dbname>) to the name you have chosen.

6. Add additional Oracle parameters or other settings as required.

   FDM has required values for the following parameters in the init<dbname>.ora. Refer to the Required Parameters for OFSA for details:

   - COMPATIBLE

     Minimum value 8.1.6.

   - DML_LOCKS

     Minimum value 200.

   - MAX_ENABLED_ROLES

     Minimum value 60.

   - OPEN_CURSORS

     Minimum value 500.

7. Save the file.

8. Create a symbolic link in the **$ORACLE_HOME/dbs** directory to the init<dbname>.ora file by entering the following information:

   ```
   ln -s $ORACLE_BASE/admin/<dbname>/pfile/init<dbname>.ora
   $ORACLE_HOME/dbs/init<dbname>.ora
   ```

Creating the FDM Database

Create the FDM database instance by either modifying the creation scripts included with the Oracle installation or modifying the script included with the FDM installation. The steps to create an FDM database are described as follows:

1. Modifying the FDM Database Scripts
2. Creating the Oracle Java VM
3. Creating the FDM Schema
4. Completing the Procedure

Modifying the FDM Database Scripts

The cr_db.sql script is in the following location: OFSA_INSTALL / dbs / <OFSA release> / master**.** If you use this script, you need to modify the following items:

Script Component	Modification
Dbname	Rename to the name you have chosen for the database.
Redo log files	Identify the name, size and location of each of these files.
Tablespaces	Create additional tablespaces, as needed.
Datafiles	Identify the name, size and location of the datafile(s) for each tablespace.
Rollback segments	Identify the number of rollback segments and the names and sizes for each, and set the optimal size appropriately.

Before you begin creating the FDM database instance verify that your environment variable, $ORACLE_SID, points to the correct instance.

Go to the $ORACLE_BASE / admin / <dbname> / create directory and complete the following steps to create the database instance.

1. Log in as a valid UNIX account that is part of the DBA group, or log in as the following UNIX account: oracle.
2. Enter the following command to start the Server Manager utility:

   ```
   svrmgrl
   ```

3. Connect to the Oracle database using the following command.

   ```
   connect internal;
   ```

4. Start the database without mounting it, using the following command.

   ```
   startup nomount;
   ```

5. Spool to the following file using the following command.

```
spool cr_db.log
```

6. Execute the following create script.

```
@OFSA_INSTALL/dbs/<OFSA release>/master/cr_db.sql
```

Creating the Oracle Java VM

FDM requires that the database instance is initialized for the Java VM. To complete this initialization, you must load the **initjvm.sql** package into your database before running any of the FDM scripts to create the database schema.

The Oracle Java VM package is found in the following location:

$ORACLE_HOME/javavm/install

To load this package into the database, go to the directory location and login to SQL*Plus as the *SYS* user. Then type the following:

```
<SQL> spool initjvm.log
<SQL> @initjvm.sql
```

Review the designated log file after the script is complete for any errors.

Creating the FDM Schema

The following topics are described in this section:

- Functional Currency
- Running the Install Procedure

Functional Currency

The FDM database creation procedure prompts for a Functional Currency. Functional Currency is defined as “the currency of the primary economic environment in which an entity conducts its business”. In a single currency environment, you need only specify your currency code. In a multiple currency environment, specify the currency used by the parent organization for financial statement reporting.

The FDM database creation process sets all tables with the ISO_CURRENCY_CD column to default to the specified Functional Currency whenever a value is not explicitly designated during an insert.

Running the Install Procedure

Before you begin creating the FDM schema verify that your environment variable, $ORACLE_SID, points to the correct instance.

Go to the **OFSA_INSTALL/dbs/<OFSA release>** directory and complete the steps in this section to create the OFSA schema. This path is referred to as the *Install Home Directory* in this chapter.

The **master/create.sql** script file (generated by the generate_install.sql script) contains the CREATE TABLE and CREATE INDEX statements for all of the OFSA tables and indexes. It is run from the **master/install script** in the steps that follow.

> **Note:** Every table and index definition in the create.sql script has the same default storage parameters. You should modify these parameters to meet your tablespace schema and sizing expectations

> **Note:** Included with the FDM data model are default Instrument and Account tables. The FDM database creation process does not create default tables that have the same name as existing user-defined objects within the FDM Schema. Refer to the *Oracle Financial Data Manager Data Dictionary* for information on the default Instrument and Account tables provided with the FDM data model.

1. In UNIX, change directory to OFSA_INSTALL / dbs / <OFSA release>.
2. Log in to the SQL*PLUS utility as SYS by entering the following command line entry:

   ```
   sqlplus sys/<password>
   ```

   Note that when logging into SQL*Plus you must do so as a local connection. Using the syntax designates a local connection to the database specified in the ORACLE_SID environment variable. This is in contrast to logging into SQL*Plus using the @ option to specify the database. This requirement exists because the Database Creation scripts create additional local database connections during execution.
3. Run the cr_owner.sql script by entering the following command line entry:

   ```
   @master/cr_owner
   ```

When prompted, enter the username you have chosen for your OFSA schema. The cr_owner.sql script creates the schema with the password that is the same as the username. Change the password after you have completed the steps to create the schema.

Caution: FDM requires that the database owner schema name for the FDM database objects be any name other than OFSA. The database owner is the schema name that owns all of the tables and other database objects used by the OFS applications. This schema name cannot be OFSA because of a naming conflict with Oracle roles required by the OFS applications.

4. Connect to the database as the owner by entering the following command line entry:

   ```
   connect <ownername>/<password>
   ```

5. Run the check_dcp script using the following command line entry:

   ```
   SQL > @master/check_dcp <Password> <Install home dir> <Sql*Loader executable>
   ```

   <Password> This is the password for the database owner, to invoke SQL*Loader.

   <Install home dir> This is the full path to the database Install home directory as previously described. Do not enter a trailing / character.

   An example of a full path follows:

   ```
   /app/ofsa/ofsa4.0-092/dbs/450001010
   ```

   <Sql*Loader executable> This is the command used to execute SQL*Loader. For most Oracle installations, this command is sqlldr.

6. Review the check_dcp.log file for any errors. You can ignore any errors relating to dropping an object. You must resolve any other errors reported in this log file before proceeding with the database creation process.

7. Run the generate_install script to generate the installation file.

   ```
   SQL > @master/generate_install <Password> <Install home dir> <Sql*Loader executable>
   ```

<Password> This is the password for the database owner, to invoke SQL*Loader.

<Install home dir> This is the full path to the database Install home directory as previously described. Do not enter a trailing / character.

An example of a full path follows:

```
/app/ofsa/ofsa4.0-092/dbs/450001010
```

<Sql*Loader executable> This is the command used to execute SQL*Loader. For most Oracle installations, this command is sqlldr.

The generate_install script then prompts for the following:

Functional Currency

The Functional Currency is the base currency for any financial statements created from the FDM database. For a list of valid Functional Currencies, refer to Appendix A, "Functional Currencies".

After you select your Functional Currency, the generate_install script prompts for confirmation:

```
You are about to install the following product(s):
OFS Applications


Do you want to proceed with this installation (y/n) >
```

Type y at the prompt to continue.

8. If you typed y to continue, the generate_install script creates a log file. Review the generate_install.log file to verify that there are no significant errors. Acceptable errors include any messages for attempting to drop objects.
9. If you typed y to continue, the generate_install script creates a **/master/create.sql**. This script contains the DDL statements for creating the FDM schema. You can modify the storage parameters and tablespace names specified in this script for creating the FDM objects.

   Caution: Every table and index definition in the create.sql script has the same default storage parameters. Modify these parameters to meet your tablespace schema and sizing expectations

10. Run the install script to create the FDM database.

```
SQL > @master/install <Password> <Install home dir> <Sql*Loader
executable>
```

<Password> This is the password for the database owner, to invoke SQL*Loader.

<Install home dir> This is the full path to the database Install home directory as previously described. Do not enter a trailing / character.

An example of a full path follows:

```
/app/ofsa/ofsa4.0-092/dbs/450001010
```

<Sql*Loader executable> This is the command used to execute SQL*Loader. For most Oracle installations, this command is sqlldr.

11. Review the following log files for any errors that may have occurred during the Database Creation Process:

- **<xxxxxxxxx>_install.log**
- **<xxxxxxxxx>_install_part_two.log**
- **generate_create_part_two.log**

where xxxxxxxxx is the FDM database release. For example: 450000037.log and 450000037_part_two.log.

You can ignore the following types of errors:

- Creation errors for public synonyms caused by *name is already used by an existing object*.
- Compilation errors for the ITG_RESOLVE_RANGE function. This function is used for integration with the Oracle General Ledger product and is therefore only operational when linked to an Oracle General Ledger database.
- Oracle error 4054 - *database link OTBFS does not exist* during the adding of the package body for the OFSA_OMM_TO_VFS package. This package is used for the integration with the Oracle Telebusiness product, and is therefore only operational when linked to an Oracle Telebusiness database.

Completing the Procedure

The Oracle installation process creates three files. The location and function of each of these files is described in the following table.

File Name	Function
/var/opt/oracle/oratab (**/etc/oratab** on some systems)	Describes the home information for each database instance. This file must have a line showing the database name, home directory and boot-up status. This file is used primarily with the following Oracle commands: dbstart and dbshut.
$ORACLE_HOME/Network/Admin or **/var/opt/oracle/listener.ora (/etc/listener.ora on some systems)**	Describes the TNS listeners connecting users to the database instances.
$ORACLE_HOME/Network/Admin or **/var/opt/oracle/tnsnames.ora** (**/etc/tnsnames.ora** on some systems)	Provides connect descriptors mapped to service names.

Finalize the installation process by completing the following tasks.

1. If you plan to use the Oracle utilities **dbshut** or **dbstart** edit the **/var/opt/oracle/oratab** or **/etc/oratab** (as appropriate for your system) with the following information:

   $ORACLE_SID:$ORACLE_HOME:BOOT_STATUS

   An example of an edited file follows:

   MYDBASE:/db/d00/app/oracle/product/8.0.4:Y

2. Configure NET8 using the following UNIX utilities provided in the $ORACLE_HOME/bin file: net8asst.sh and/or net8wiz.sh. Be sure to consult the *NET8 Administrator's Guide* to configure NET8.

   After configuring NET8 you should have the following two files:

   - listener.ora
   - tnsnames.ora

3. Start NET8 on the server by using the **lsnrctl** command in the $ORACLE_HOME/bin file.

An example of this lsnrctl command follows:

```
lsnrctl start LISTENER_NAME
```

4. Verify that the listener is running and configured correctly using the lsnrctl command.

   An example of this lsnrctl command follows:

   ```
   lsnrctl stat LISTENER_NAME
   ```

5. Test the listener against your database using the command example that follows:

   ```
   sqlplus schema/password@tnsnames_entry
   ```

11

Upgrading from OFSA 3.5/4.0

This chapter provides information for users upgrading from the Oracle Financial Services Applications (OFSA) database versions 3.5 and 4.0 to Release 4.5 of the Oracle Financial Data Manager (FDM) database. The FDM Database Upgrade process migrates existing OFSA 3.5 and 4.0 database structures to the new FDM 4.5 database environment. This chapter describes how this migration process affects client data database structures.

The following specific migration topics are covered in this chapter:

- Rename of FDM Reserved Objects to OFSA_
- Portfolio Instrument and Services Tables
- Non-Portfolio Instrument Tables
- LEDGER_STAT
- Code Descriptions
- Multiprocessing Settings
- Financial Elements
- PROCESS_CASH_FLOWS
- Collateral Data Model
- ID Conversions

Note: Information regarding how the 4.5 upgrade process affects security is discussed in the Chapter 14, "FDM Security".

Note: The Migration procedure registers only FDM 4.5 Metadata for those objects that are identified in the OFSA 3.5/4.0 SYSTEM_INFO table and which exist as valid tables or views in the Oracle RDBMS. The 4.5 Migration procedure does not convert metadata for synonyms or views in the OFSA 3.5/4.0 database that point to objects of a different name.

Rename of FDM Reserved Objects to OFSA_

All internal application objects, such as those used to store the OFSA IDs, are renamed in FDM 4.5 with an OFSA_ prefix. Any table with such a prefix is designated as an FDM Reserved Object.

The FDM database upgrade process automatically updates OFSA IDs to reference the new object names. However, the FDM database upgrade process does not update SQL IDs, custom stored procedures or custom Discoverer Reports. After the upgrade process is complete, you must update any of these that reference FDM Reserved Objects.

Portfolio Instrument and Services Tables

The term *Portfolio Instrument table* is used to refer to tables storing financial account data for use with the OFS applications. The term *Services* table designates the account services tables such as CC and CD.

Tables in OFSA 3.5/4.0 with a SYSTEM_INFO data_code value in (2,6,8) are considered to be in the Portfolio Instrument and Services Tables category for purposes of the FDM 4.5 Database Upgrade. To identify the tables in this category, run the following SQL statement:

```
SELECT table_name FROM system_info S, all_tables A
WHERE S.data_code in (2,6,8)
AND A.owner = :fdm_schema_owner
AND S.table_name = A.table_name;
```

Any table in this list is affected by the upgrade as follows:

- Multi-Currency Enablement
- TP Option Costing

- Table Classification Validation

Multi-Currency Enablement

FDM 4.5 supports a multi-currency environment. To this end, the FDM 4.5 Database Upgrade process adds a new column, ISO_CURRENCY_CD, to all OFSA 3.5 and 4.0 Portfolio Instrument and Services tables. The Database Upgrade process also updates this column with a default currency value for all instrument records based upon the Functional Currency specified during the Metadata Migration phase. When you launch the Migration phase of the Database Upgrade Process, it prompts for a Functional Currency Code, which is then used for this update.

For tables with a large number of records, the addition of this new column and the update of the existing instrument records can require significant time to complete. The storage definition for instrument tables can also be affected by this update.

Note: Previous versions of OFSA included a CURRENCY_CD column on Instrument tables for reporting purposes. ISO_CURRENCY_CD column is now used to designate the Currency in FDM 4.5. FDM preserves the CURRENCY_CD column on all instrument tables so that you can map values from this column to the new ISO_CURRENCY_CD standard using the Currency Mapping utility described in Chapter 21, "FDM Utilities".

TP Option Cost Calculations

The Database Upgrade process adds the following fields to all Portfolio Instrument and Services tables:

Column Name	Column Definition
ORG_MARKET_VALUE	NUMBER(14,2)
CUR_OAS	NUMBER(8,4)
CUR_STATIC_SPREAD	NUMBER(8,4)
HISTORIC_OAS	NUMBER(8,4)
HISTORIC_STATIC_SPREAD	NUMBER(8,4)

Table Classification Validation

The Database Upgrade process assigns Table Classifications to Portfolio Instrument and Services tables so that they are available within FDM for OFS processing operations. The Database Upgrade process then validates these assignments and reports any Table Classification assignment failures in the part_two upgrade log file. Note that this is legitimate for a table to fail a Table Classification assignment. OFSA 3.5 and 4.0 supported the flexible columns implementation, which means that some tables may not have all of the required columns to satisfy a particular Table Classification assignment.

All Instrument and Services tables from OFSA 3.5/4.0 are assigned to one or more of the following Table Classifications:

Table Classification CD	Table Classification
20	Instrument
200	TP Cash Flow
210	TP Non-Cash Flow
310	Instrument Profitability
360	RM Standard
330	Data Correction Processing
370	TP Option Costing

The Database Upgrade process validates these Table Classification assignments and reports any assignment failures to the **<xxxxxxxxx>_part_two.log**, where xxxxxxxxx is the FDM database release. Refer to Chapter 12, "FDM Database Upgrade Process" for more information regarding reviewing the upgrade log files. Refer to Chapter 16, "FDM Object Management" for more information on the Table Classification requirements.

Non-Portfolio Instrument Tables

Non-Portfolio Instrument tables are tables used in Oracle Performance Analyzer Allocation processing that do not possess the Oracle Transfer Pricing or Oracle Risk Manager columns. This category includes, but is not limited to, tables storing account transaction data.

Transaction tables (also referred to in OFSA 3.5/4.0 as Vertical Instrument Activity Tables, or VIAT) are a subclass of Non-Portfolio Instrument tables. These tables are only unofficially supported within the OFSA 3.5/4.0 metadata. As such, the FDM 4.5 Database Upgrade process has no way of identifying transaction tables in an OFSA 3.5/4.0 database.

Transaction tables in OFSA 3.5/4.0 are tables with a DATA_CODE assignment of *3* and a unique index consisting of ID_NUMBER, IDENTITY_CODE and one or more Leaf columns. The database upgrade process treats these tables as Non-Portfolio Instrument tables.

Tables in OFSA 3.5/4.0 with a SYSTEM_INFO data_code value = 3 are considered to be in the Non-Portfolio Instrument Table category for purposes of the FDM 4.5 Database Upgrade. To identify the tables in this category, run the following SQL statement:

```
SELECT table_name FROM system_info S, all_tables A
WHERE S.data_code = 3;
AND A.owner = :fdm_schema_owner
AND S.table_name = A.table_name;
```

Any table in this list is affected by the upgrade as follows:

- Multi-Currency Enablement
- Table Classification Validation

Multi-Currency Enablement

FDM 4.5 supports a multi-currency environment. To this end, the FDM 4.5 Database Upgrade process adds a new column, ISO_CURRENCY_CD, to all OFSA 3.5 and 4.0 Non-Portfolio Instrument tables. The Database Upgrade process also updates this column with a default currency value for all instrument records based upon the Functional Currency specified during the Metadata Migration phase. When you launch the Migration phase of the Database Upgrade Process, it prompts for a Functional Currency Code, which is then used for this update.

For tables with a large number of records, the addition of this new column and the update of the existing instrument records can require significant time to complete. The storage definition for instrument tables can also be affected by this update.

Table Classification Validation

The Database Upgrade process assigns Table Classifications to Non-Portfolio Instrument tables so that they are available within FDM for OFS processing

operations. The Database Upgrade process then validates these assignments and reports any Table Classification assignment failures in the part_two upgrade log file. Note that this is legitimate for a table to fail a Table Classification assignment. OFSA 3.5 and 4.0 supported the flexible columns implementation, which means that some tables may not have all of the required columns to satisfy a particular Table Classification assignment.

All Non-Portfolio Instrument tables from OFSA 3.5/4.0 are assigned to one or more of the following Table Classifications:

Table Classification CD	Table Classification
20	Instrument
310	Instrument Profitability
330	Data Correction Processing

The Database Upgrade process validates these Table Classification assignments and reports any assignment failures to the **<xxxxxxxxx>_part_two.log**, where xxxxxxxxx is the FDM database release. Refer to Chapter 12, "FDM Database Upgrade Process" for more information regarding reviewing the upgrade log files. Refer to Chapter 16, "FDM Object Management" for more information on the Table Classification requirements.

> **Note:** After the database upgrade process is complete, manually assign any Transaction (VIAT) tables to the Transaction Profitability Table Classification. For these tables, you should also revoke the Instrument Profitability classification that the database upgrade process assigns automatically. The FDM 4.5 database upgrade process has no way of identifying which OFSA 3.5/4.0 Non-Portfolio Instrument tables are actually Transaction tables.

LEDGER_STAT

The LEDGER_STAT table is an FDM reserved table that stores client data. The LEDGER_STAT table is affected by the upgrade as follows:

- Multi-Currency Enablement

Multi-Currency Enablement

FDM 4.5 supports a multi-currency environment. To this end, the FDM 4.5 Database Upgrade process adds a new column, ISO_CURRENCY_CD, to the LEDGER_STAT table. The Database Upgrade process also updates this column with a default currency value for all instrument records based upon the Functional Currency specified during the Metadata Migration phase. When you launch the Migration phase of the Database Upgrade Process, it prompts for a Functional Currency Code, which is then used for this update.

For tables with a large number of records, the addition of this new column and the update of the existing instrument records can require significant time to complete. The storage definition for instrument tables can also be affected by this update.

Code Descriptions

The OFSA 3.5/4.0 database stores names and descriptions for Code values in the SYSTEM_CODE_VALUES table. FDM 4.5 migrates the data stored in this table to individual data structures for each Code column.

The migration of this data is dependent upon the type of Code column. There are four types of Code columns in FDM:

- FDM Reserved Codes
- User Editable Codes
- User Defined Codes
- Interest Rate Codes
- Product Type Code

This section discusses the impact of each of these migrations.

FDM Reserved Codes

FDM Reserved Codes are code columns for which FDM reserves the entire range of values. FDM 4.5 creates new data structures for each individual FDM Reserved Code column and seeds these data structures with the acceptable set of values. Data from SYSTEM_CODE_VALUES is not migrated to the FDM 4.5 database. If users have created any new code values or changed existing code descriptions for these code columns in OFSA 3.5/4.0, these changes are removed by the 4.5 database upgrade process.

Code Descriptions for FDM Reserved Code columns are MLS enabled. FDM 4.5 creates the following objects for each Code column:

- Base Table (named _CD)
- MLS Table (named _MLS)
- Language Compatible View (named _DSC)

The upgrade process also creates the appropriate triggers for these objects to maintain the Multi-Language Support functionality. For more information about these objects, and about multi-language support within FDM, refer to Chapter 15, "FDM Multi-Language Support".

Note: Only a subset of the Code columns are available for use with Instrument and Client Data Objects. The remaining Code columns are reserved for use with FDM internal application objects. Refer to the *Oracle Financial Data Manager Data Dictionary* for information on Code columns available for use with Instrument and Client Data Objects.

The following code columns are reserved by FDM:

FDM Reserved Code Columns

ACCRUAL_BASIS_CD	OVERDRAFT_PROTECTION_CD
ACCUMULATION_TYPE_CD	OWNERSHIP_CD
AMOUNT_TYPE_CD	PATTERN_TYPE_CD
ASSIGNMENT_DATE_CD	PAYMENT_PATTERN_TYPE_CD
ASSIGNMENT_METHOD_CD	PAYMENT_TYPE_CD
BATCH_EVENT_STATUS_CD	RCV_ACCRUAL_BASIS_CD
BATCH_EVENT_TYPE_CD	RCV_COMPOUND_BASIS_CD
BOOKING_CD	PMT_TYPE_CD
CALC_METHOD_CD	PP_DIM_TYPE_CD
CALC_MODE_CD	PRIVATE_MORTGAGE_INSURER_CD
CALC_SOURCE_CD	PROCESS_FILTER_TYPE_CD
COLUMN_PROPERTY_CD	PROCESS_PARTITION_CD

COMPONENT_TYPE_CD	PROCESS_SCOPE_CD
COMPOUND_BASIS_CD	PUT_CALL_CD
CONFORMANCE_CD	QUERY_ROLE_CD
CONTACT_METHOD_CD	QUOTE_CD
CURRENCY_STATUS_CD	RATE_CAP_TYPE_CD
DETAIL_RECORD_CD	RATE_CHG_RND_CD
DIMENSION_TYPE_CD	RATE_CHG_ROUNDING_CD
DISCOUNT_RATE_METHOD_CD	RATE_DATA_SOURCE_CD
ESTIMATION_SMOOTHING_CD	RATE_FLOOR_TYPE_CD
EXCHANGE_RATE_CONVERT_TYPE_CD	RATE_TERM_CD
EXCHANGE_RATE_STATUS_CD	PAY_ACCRUAL_BASIS_CD
FBAL_METHOD_CD	PAY_COMPOUND_BASIS_CD
FCAST_IRC_METHOD_CD	RATE_VOLUME_REL_CD
FCAST_XRATE_METHOD_CD	REGULATION_D_STATUS_CD
FINANCIAL_SCENARIO_CD	REPRICE_METHOD_CD
FORWARD_TYPE_CD	RESP_PARTY_CD
FREQUENCY_UNIT_CD	RESULT_TYPE_CD
ID_TYPE_CD	ROLL_FACILITY_CD
INCENTIVE_TYPE_CD	RUNOFF_TYPE_CD
INT_TYPE	SETTLEMENT_TYPE_CD
INTEREST_COMPONENT_TYPE_CD	SMOOTHING_METHOD_CD
INTEREST_TIMING_TYPE_CD	SOURCE_CD
IRC_FORMAT_CD	SOURCE_TBL_TYPE_CD
ISO_CURRENCY_CD	STOCH_RANDOM_SEQ_TYPE_CD
JOB_STATUS_CD	STRIKE_TYPE_CD
LEAF_DATA_SOURCE_CD	STRING_CD
MEASURE_CD	TABLE_CLASSIFICATION_CD
MESSAGE_CD	TARGET_BAL_CD
MODIFY_ACTION_CD	TERM_TYPE_CD

MSG_SEVERITY_CD	TM_PROCESS_TYPE_CD
MULTIPLIER_CD	TP_CALC_METHOD_CD
NET_MARGIN_CD	TRACKING_METHOD_CD
OD_PROTECTION_CD	TRACKING_STATUS_CD
OPTION_COST_METHOD_CD	TS_MODEL_CD
OPTION_EXERCISE_CD	USAGE_CD
OPTION_TYPE_CD	

User Editable Codes

Code columns seeded by FDM but for which FDM enables users to create new code values are designated as User Editable Codes. For each of these code columns, FDM seeds MLS enabled data structures as well as a reserved range of code values. Users are then allowed to add new code description entries to these data structures.

FDM 4.5 creates new MLS enabled data structures for each individual FDM User Editable Code column and seeds the reserved range for these data structures. The upgrade process also migrates any user entered code values for these Code columns from the OFSA 3.5 / 4.0 SYSTEM_CODE_VALUES table into the new data structures. However, any user entered code values in the reserved range are deleted during this process.

Code Descriptions for User Editable Code columns are MLS enabled. FDM 4.5 creates the following objects for each Code column:

- Base Table (named _CD)
- MLS Table (named _MLS)
- Language Compatible View (named _DSC)

The upgrade process also creates the appropriate triggers for these objects to maintain the Multi-Language Support functionality. For more information about these objects, and about Multi-Language Support within FDM, refer toChapter 15, "FDM Multi-Language Support".

Note: While the PRODUCT_TYPE_CD column is designated as a User Editable Code, the migration of the code descriptions for this column is a special case for FDM 4.5. Refer to Product Type Code for more information.

The following code columns are designated as User Editable:

User Editable Code Columns

ADJUSTABLE_TYPE_CD	INSTRUMENT_TYPE_CD
AGENCY_CD	INSURANCE_TYPE_CD
AMORTIZATION_TYPE_CD	ISSUER_CD
AMRT_TYPE_CD	LIEN_POSITION_CD
CMO_TRANCHE_CD	LIQUIDITY_CLASS_CD
COLLATERAL_CD	LOAN_PROPERTY_TYPE_CD
COLLATERAL_STATUS_CD	MARKET_SEGMENT_CD
COLLATERAL_SUB_TYPE_CD	MORTGAGE_AGENCY_CD
COLLATERAL_TYPE_CD	OCCUPANCY_CD
COMMITMENT_TYPE_CD	PAY_ADJUSTABLE_TYPE_CD
CONSOLIDATION_CD	PLEDGED_STATUS_CD
CREDIT_RATING_CD	PRODUCT_TYPE_CD
CREDIT_STATUS_CD	PROPERTY_TYPE_CD
DIRECT_IND_CD	PURPOSE_CD
DISCHARGE_TYPE_CD	RELATIONSHIP_CD
ERROR_CODE	RCV_ADJUSTABLE_TYPE_CD
EXIST_BORROWER_CD	SERVICING_AGENT_CD
GEOGRAPHIC_LOC_CD	SIC_CD
HELD_FOR_SALE_CD	SOLICIT_SOURCE_CD

User Defined Codes

Code columns created by users are designated as User-Defined. Such columns are not part of the standard FDM data model. Rather, they are new columns added by users to the OFSA 3.5/4.0 database. OFSA 3.5/4.0 stores the descriptions for these columns in the SYSTEM_CODE_VALUES table.

The FDM 4.5 database upgrade process creates non-MLS enabled table structures to store values for these codes. Unlike the seeded code columns, each User Defined

code column is migrated into a single individual Description table. This table supports code descriptions in a single language only.

If the Code column is designated in SYSTEM_CODE_VALUES for the INSTRUMENT='ALL', FDM migrates the descriptions by creating an individual Description table for each distinct code column. The table name is prefixed with the code column name, and is suffixed with a _DSC extension to designate that it stores the code descriptions for that column. Long code column names will result in a truncation in the table name so that it meets the Oracle RDBMS limit of 30 characters.

If the Code column is designated in SYSTEM_CODE_VALUES for INSTRUMENT <>'ALL', then the code descriptions apply only to the specific tables listed. In this case, the FDM database upgrade creates a separate Description table for each unique combination of code COLUMN_NAME and INSTRUMENT.

Interest Rate Codes

The Interest Rate Code columns (and its alias column names) are a special case. Values for these columns are created using Oracle Financial Data Manager Rate Manager, rather than being entered directly into a Code table.

FDM designates the following as Interest Rate Code columns:

- INTEREST_RATE_CD
- PAY_INTEREST_RATE_CD
- RCV_INTEREST_RATE_CD
- STRIKE_INTEREST_RATE_CD

Product Type Code

The PRODUCT_TYPE_CODE column is designated as a User Editable code column. However, the seeded Product Type codes and code descriptions in FDM 4.5 are significantly changed from the seeded values for OFSA 3.5/4.0. Because of this, the migration of code descriptions for the PRODUCT_TYPE_CODE column is a special case for FDM 4.5.

The OFSA 3.5/4.0 database seeded a set of PRODUCT_TYPE_CODE values for each distinct Instrument table name. For example, the seeded set of Product Types for the DEPOSITS table would be a different set than the seeded Product Types for COMMERCIAL_LOAN. The codes overlapped in many cases, with the same

numeric PRODUCT_TYPE_CD associated with different descriptions in each Instrument table.

In FDM 4.5, all of the seeded Product Type codes are consolidated into a single set that is valid for all Instrument and Account tables. FDM stores Product Type Code descriptions in a single set of data structures that is MLS enabled. The seeded data structures are:

OFSA_PRODUCT_TYPE_CD (Base table)

OFSA_PRODUCT_TYPE_MLS (MLS table)

OFSA_PRODUCT_TYPE_DSC (Language Compatible View)

FDM still designates this column as User Editable, so that users are allowed to enter their own Product Type codes and descriptions as needed.

The *old* Product Type Codes previously stored in SYSTEM_CODE_VALUES are retained, however, FDM migrates them as User Defined code descriptions and creates individual, non-MLS enabled Description tables for each. Because a separate set of these codes were originally seeded for each distinct Instrument table, the FDM database upgrade process creates a separate Product Type Code description table for each such set recorded in SYSTEM_CODE_VALUES. The upgrade process maps any existing instrument tables to these individual Product Type Code tables in OFSA_DESCRIPTION_TABLES.

Thus, all of the Product Type Codes from OFSA 3.5/4.0 are preserved as User Defined Code tables. The new data structures for the seeded Product Type Codes are also created for your database. If you decide to use the new, MLS enabled data structures for Product Type Code, you need to complete the following:

- Modify data extracts to use the new Product Type Codes
- Edit the Description Table Mappings (which can be done in the FDM Administration Application) for the PRODUCT_TYPE_CD column on all existing Instrument tables to map to the new OFSA_PRODUCT_TYPE_DSC view

Because the *old* Product Type Codes had overlapping code values, you need to modify your data extract routines to populate this column with the set of Product Type Codes available in OFSA_PRODUCT_TYPE_DSC.

If you do decide to migrate to the new Product Type data structures, you are free to unregister and drop any of the User Defined Product Type Description tables created by the 4.5 FDM database upgrade process.

Multiprocessing Settings

OFS application multiprocessing settings in FDM 4.5 are no longer specified in the server ini files. Rather, they are designated in the database. Because of this, all OFS application multiprocessing settings revert to the default after the FDM upgrade process is complete. Refer to Chapter 19, "OFSA Multiprocessing" for more information

Financial Elements

FDM introduces the categorization of Financial Elements for Transformation processing purposes. Financial Elements are specific values for the FINANCIAL_ELEM_ID Leaf Column. Transformation processing requires that Financial Elements are assigned to one of the following categories:

- Balance
- Balance Weighted Object
- Standard Rate
- Statistic

FDM seeded Financial Elements are assigned to one of these categories during the FDM database upgrade process. The FDM upgrade process also categorizes all user defined Financial Elements as Statistic. FDM designates the category by assigning a value to the Column Property field for each leaf.

Edit or view Column Property assignments for user defined Financial Elements from Leaf Setup within the OFS Applications.

PROCESS_CASH_FLOWS

The 4.5 FDM database upgrade process does not convert data from the OFSA 3.5/4.0 PROCESS_CASH_FLOWS table to the new table OFSA_PROCESS_CASH_FLOWS. Because the data in this table is for the auditing of Risk Manager and Transfer Pricing process runs, it is not necessary to migrate this data to the new table structure. Re-running the required Risk Manager or Transfer Pricing Process IDs regenerates data for the new OFSA_PROCESS_CASH_FLOW table.

The upgrade process saves the data from OFSA 3.5/4.0 PROCESS_CASH_FLOWS table into a table named O_PROCESS_CASH_FLOWS.

Collateral Data Model

FDM 4.5 introduces a new set of objects supporting Collateral information. These Collateral objects are considered part of the FDM standard data model. The FDM upgrade process automatically creates these objects.

Prior to running the FDM database upgrade procedure, verify that you do not already have these object names in your FDM Schema. In order for these objects to be created properly, the FDM database upgrade process requires that they do not exist in the schema.

The new Collateral objects are:

Table Name
ACCOUNT_COLLATERAL
ACCOUNT_GUARANTOR_RELATION
COLLATERAL
COLLATERAL_ASSESSMENT_HISTORY
COLLATERAL_AUCTION_DETAILS
COLLATERAL_BOATS
COLLATERAL_INSURANCE_DETAILS
COLLATERAL_OTHER_INSTITUTIONS
COLLATERAL_OWNERS
COLLATERAL_REAL_ESTATE
COLLATERAL_SHARES
COLLATERAL_VEHICLES

ID Conversions

As a result of the introduction of multi-currency functionality in the 4.5 release, as well as other functionality enhancements for Rate Manager, Risk Manager, Transfer Pricing, and Performance Analyzer, the underlying data structure of some OFSA IDs has changed. The FDM database upgrade process converts information stored for these IDs to the new 4.5 database structure.

Where functionality has significantly changed, the conversion routines make assumptions as to how the data is converted. Conversion routines support only upgrades from OFSA 3.5 and OFSA 4.0. The converted IDs may not generate the same results as the old IDs. Before you process a newly converted database, it is recommended that you review the converted IDs for content and update them where necessary.

This chapter describes the conversion routines for the following IDs:

- Allocation ID
- Configuration ID
- Discount Rates ID
- Forecast Balance ID
- Forecast Rates ID
- Historical Rates ID
- Leaf Characteristics ID
- Maturity Strategy ID
- Pricing Margin ID
- RM Process ID
- Term Structure ID
- TP Process ID
- Transaction Strategy ID
- Transfer Pricing ID

Allocation ID

The FDM 4.5 Database Upgrade process converts Allocation IDs from OFSA 3.5/4.0 to the FDM 4.5. This conversion includes the following:

- Add Missing Leaf Columns
- Remove Extraneous Leaf Columns
- Mirror to Table Update
- Error Messages

Note: Performance Analyzer 4.5 employs operator precedence logic that is different from that employed in version 3.5/4.0. Review the alloc_id_conv.log for information regarding Allocation IDs that may be impacted by this logic change. The alloc_id_conv.log also provides information about the results of the Allocation ID conversion routine.

Add Missing Leaf Columns

If there is a Leaf Column identified in OFSA_CATALOG_OF_LEAVES where TABLE_TYPE<> 'V' missing from the Allocation ID, the Allocation conversion routine performs the following logic:

Debit (ALLOC_LEAVES.ROW_TYPE=10) and Credit (ALLOC_LEAVES.ROW_TYPE=20) rows:
For each Allocation ID Page Loop:

For each Product Leaf Column in OFSA_CATALOG_OF_LEAVES where OFSA_CATALOG_OF_LEAVES.TABLE_TYPE<>'W' and Leaf Column is missing Debit/Credit rows:

- IF ALLOC_LEAVES.LEAF_NUM_ID=0 (Financial_elem_id) and ALLOC_LEAVES.LEAF_NODE=-99100 (None) then set OFSA_ALLOC_LEAVES.LEAF_NODE for the missing Leaf Column to -99100 else -99300(Same As Filter).
- OFSA_ALLOC_LEAVES.TREE_SYS_ID=0
- OFSA_ALLOC_LEAVES.TREE_LEVEL_NUM = 0
- OFSA_ALLOC_LEAVES.FIN_ELEM_CD = 0
- OFSA_ALLOC_LEAVES.MIRROR_TO_TABLE = 'LEDGER_STAT'
- The rest of the columns have the same value as the other Product Leaf Columns.

Filter(ALLOC_LEAVES.ROW_TYPE=30) and Percentage (ALLOC_LEAVES.ROW_TYPE=40,45)
For each Allocation ID Page Loop

For each Product Leaf Column in OFSA_CATALOG_LEAVES where OFSA_CATALOG_LEAVES.TABLE_TYPE <> 'V' and Leaf Column is missing Filter/Percentage rows:

- IF the missing Product Leaf Column exists in the table that you wanted to allocate then OFSA_ALLOC_LEAVES.LEAF_NODE= -99200 (ALL) else -99100(None)
- FSA_ALLOC_LEAVES.TREE_SYS_ID=0
- FSA_ALLOC_LEAVES.TREE_LEVEL_NUM = 0
- FSA_ALLOC_LEAVES.FIN_ELEM_CD = 0
- FSA_ALLOC_LEAVES.MIRROR_TO_TABLE = 'LEDGER_STAT'
- The rest of the columns have the same value as the other Product Leaf Columns.

Remove Extraneous Leaf Columns

When any of the Product Leaf Columns exist in OFSA_ALLOC_LEAVES and NOT IN OFSA_CATALOG_OF_LEAVES then the Product Leaf Columns rows are removed from OFSA_ALLOC_LEAVES.

(Only applies to OFSA_ALLOC_LEAVES.ROW_TYPE = 10,20,30,40,50)

Mirror to Table Update

Update OFSA_ALLOC_LEAVES set MIRROR_TO_TABLE = 'LEDGER_STAT';

Error Messages

There are some conditions for which the Allocation conversion is unable to set the Leaf Column value. Such conditions may cause an error when the Allocation ID is opened within Performance Analyzer 4.5:

1. Allocation IDs that do not have a Filter On but either the Debit or the Credit Leaf value is set to Same As Filter.
2. Allocation IDs where the Filter O table is a Detail table, but the Debit or Credit is set to Ledger_Stat and there is an *L* type leaf value set to Same As Filter.
3. Allocation IDs where the Filter On table is a Detail table but the Debit or Credit is identified with Post to Ledger_Stat set to On and there is an *L* type leaf value set to Same As Filter.

Allocation IDs with these conditions are logged to a log file named **alloc_id_ conv.log**. Review this log file at the end of the FDM 4.5 database upgrade process to identify any potential Allocation ID problems.

Configuration ID

1. Groups from OFSA 3.5/4.0 are now referred to as ID Folders for the purposes of categorizing OFSA IDs. The Default Group Name is now the Default Folder Name for Configuration IDs.
2. The Configuration ID no longer designates a Historical Rates ID for the session.
3. The Configuration ID no longer designates a Compound Basis for the session. This functionality is now part of Rate Manager.

Discount Rates ID

1. OFSA_IDT_DISCOUNT_RATE
   - Update all the values in ISO_CURRENCY_CD column to 000.
   - Update all the values in INTEREST_COMPONENT_TYPE_CD to 1.

Forecast Balance ID

If you are upgrading from OFSA 3.5, the FDM Database Upgrade Process executes the logic described in the Upgrades from OFSA 3.5 section first before executing the logic described in the All Upgrades section.

Upgrades from OFSA 3.5

The Forecast Balance ID conversion takes information from the existing Forecast Balance ID and translates the format into the new Forecast Balance ID format. The Forecast Balance methodologies are mapped from the previous release's methodology selections to one of the following new business methodologies:

- No New Business
- New Add
- Target Average
- Target End
- Target Growth
- Roll-over
- Roll-over with New Adds

A description of the assignment process, per method, is provided as follows:

New Add Methods

The conversion assignment depends on roll-into assumptions for the current product. If the current product does not have roll-into assumptions, the conversion process assigns New Add as the forecast balance method. If roll-into assumptions exist for the product, Roll-over with New Add is assigned as the forecast method.

> **Note:** Roll-into assumptions include other products rolling into the product or roll-over from the product back into itself.

In all New Add cases, the timing method is set to At Bucket End.

Target Average Method

The Target Average method does not change between releases. If roll-into assumptions exist for the product, the conversion deletes the roll-into assumptions.

Target End Method

The Target End method does not change between releases. If roll-into assumptions exist for the product, the conversion deletes the roll-into assumptions.

Roll-over Method

If roll-into assumptions exist for the product and no other new business assumptions exist, the conversion assigns the Roll-over method. Two other issues may need to be addressed during conversions of roll-over assumptions:

Bucket Range Definition

The conversion may need to employ complex logic in defining the percents per modeling bucket. In prior versions, the user defined roll-over bucket ranges for each source leaf. Since Release 4.0, the user defines bucket ranges for each target leaf. As a result, the conversion routine must create consistent bucket ranges for all source leafs generating roll-over for a single target leaf.

The following table represents source leafs for a single target leaf, 5-Year CD. Note that the bucket ranges are different for every source.

Input Data for Roll-over Conversion

Source Leaf	Bucket Range	Roll Percent
5 Year CD	1 to 12	100%
5 Year CD	13 to 120	80%
3 Year CD	1 to 12 0	60%
1 Year CD	1 to 5	40%

To convert the data, the conversion must determine the superset of bucket ranges. This is accomplished by starting with the smallest bucket range and working through all other bucket ranges, breaking up the data as necessary. The resulting input rows are displayed, as follows:

Output Data for Roll-over Conversion

Source Leaf	Bucket Range	Roll Percent
5 Year CD	1 to 5	100%
5 Year CD	6 to 12	100%
5 Year CD	13 to 120	80%
3 Year CD	1 to 5	60%
3 year CD	6 to 12	60%
3 Year CD	13 to 120	60%
1 Year CD	1 to 5	40%
1 Year CD	6 to 12	0%
1 Year CD	13 to 120	0%

Missing Buckets within the Bucket Range

The conversion routine determines the bucket range from the maximum and minimum forecasted bucket for each product leaf. If intermediate buckets are missing in this bucket range, filler buckets are generated with a balance of zero. This issue would occur if not all buckets within the range had forecast balance

assumptions. In other words, an assumption was defined for bucket 1, bucket 3 and bucket 5 with no assumptions for buckets 2 and 4.

All Upgrades

1. OFSA_IDT_FORECAST_BAL
   - Update all the values in ISO_CURRENCY_CD column to OFSA_DB_INFO.FUNCATIONAL_CURRENCY_CD
2. OFSA_FBAL_ROLL_INTO
   - Update all the values in ISO_CURRENCY_CD column to OFSA_DB_INFO.FUNCATIONAL_CURRENCY_CD.
3. OFSA_FBAL_VOLUMES
   - Update all the values in ISO_CURRENCY_CD column to OFSA_DB_INFO.FUNCATIONAL_CURRENCY_CD.
4. OFSA_FBAL_DIMENSIONS
   - Update all the values in ISO_CURRENCY_CD column to OFSA_DB_INFO.FUNCATIONAL_CURRENCY_CD.

Forecast Rates ID

The Forecast Rates ID conversion routine creates a **fcast_rate_id_conv.log** file for messages and information about the conversion.

Note: The Forecast Rates ID converts all scenarios for all interest rate codes to the method Structured Change for every bucket. This can create unnecessary excess data. The new structure of the Forecast Rates ID also records the term of each modeling bucket. Because existing Forecast Rates IDs may have been created under differing Configuration IDs and therefore differing modeling buckets, the Conversion routine assumes that each bucket is 1 month. Because of this, if a newly converted Forecast Rates ID is opened under a Configuration ID with non-monthly buckets, a warning is displayed. Once the ID is saved again, the new modeling bucket terms are saved with it. After the conversion, end-users may want to review the stored data and save the Forecast Rates ID with a more optimal data set, depending upon their desired rate change and modeling bucket definition.

The mapping logic to the new FDM 4.5 data structure is as follows:

Target Column OFSA_IDT_FORECAST_RATES	Source Column OFSA_CATALOG_OF_IDS	Special Conversion Logic
FCAST_RATES_SYS_ID	SYS_ID_NUM	Must exist in OFSA_ CATALOG_OF_IDS where ID_TYPE = 305
REPORTING_CURRENCY_CD	N/A	Plug with OFSA_DB_ INFO.Functional_Currency_ Cd.

Target Column OFSA_FCAST_RATES_SCENARIOS	Source Column RATES_SCENARIO	Special Conversion Logic
FCAST_RATES_SYS_ID	RATES_SYS_ID	FCAST_RATE_SYS_ID must exist in OFSA_IDT_ FORECAST_RATES.
SCENARIO_NUM	SCENARIO_NUM	

Target Column OFSA_FCAST_RATES_SCENARIOS	Source Column RATES_SCENARIO	Special Conversion Logic
SCENARIO_NAME	DESCRIPTION	RATES_ SCENARIO.DESCRIPTION column is 80 characters long. So, the conversion truncates the description to 40 characters. If the scenario name is a duplicate within a Forecast Rate ID, then the conversion routine adds a number suffix in the scenario name.

Target Column OFSA_FCAST_IRCS	Source Column RATES_FORECAST	Special Conversion Logic
FCAST_RATES_SYS_ID	RATES_SYS_ID	
SCENARIO_NUM	SCENARIO_NUM	
INTEREST_RATE_CD	INTEREST_RATE_CD	
FCAST_IRC_METHOD_CD	N/A	Plug 2
BASE_SCENARIO_NUM	N/A	Plug 0

Target Column OFSA_FCAST_IRC_STRCT_CHG_BKT	Source Column RATES_FORECAST	Special Conversion Logic
FCAST_RATES_SYS_ID	RATES_SYS_ID	See below
SCENARIO_NUM	SCENARIO_NUM	See below
INTEREST_RATE_CD	INTEREST_RATE_CD	See below
FROM_BKT_NUM	BUCKET	See below
TO_BKT_NUM	BUCKET	See below

The logic to create the bucket data is as follows:

1. From the table RATES_FORECAST, select the RATES_SYS_ID and MAX (RATE_FORECAST.BUCKET).
2. Within a Forecast Rate ID for each INTEREST_RATE_CD and SCENARIO_ NUM, insert an entry for each bucket in OFSA_FCAST_IRC_STRCT_CHG_BKT until it reaches the MAX(RATE_FORECAST.BUCKET).

Target Column OFSA_FCAST_IRC_STRCT_CHG_VAL	Source Column RATES_FORECAST	Special Conversion Logic
FCAST_RATES_SYS_ID	RATES_SYS_ID	See below
SCENARIO_NUM	SCENARIO_NUM	See below
INTEREST_RATE_CD	INTEREST_RATE_CD	See below
FROM_BKT_NUM	BUCKET	See below
INTEREST_RATE_TERM	INTEREST_RATE_TERM	See below
INTEREST_RATE_TERM_MULT	INTEREST_RATE_MULT	See below
RATE_CHANGE	INTEREST_RATE	See below

The conversion routine loops through all rows stored for each Rates Sys ID, Scenario, Interest Rate Code, Interest Rate Term, Interest Rate Term Multiplier and finds all explicit rate changes using the following process:

1. Order all rows from RATE_FORECAST by Rates Sys ID, Scenario, Interest Rate Code, Interest Rate Term, Interest Rate Term Multiplier and Bucket.
2. Read in the first row and set the following variables:
   - sys_ID = RATES_FORECAST. RATES_SYS_ID
   - scenario = RATES_FORECAST. SCENARIO_NUM
   - IRC = RATES_FORECAST. INTEREST_RATE_CD
   - from_bucket_num = RATES_FORECAST.BUCKET
   - rate_change = RATES_FORECAST.INTEREST_RATE - PREVIOUS BUCKET RATES_FORECAST.INTEREST_RATE
   - term = RATES_FORECAST. INTEREST_RATE_TERM
   - mult= RATES_FORECAST. INTEREST_RATE_MULT

3. Make sure all the FROM_BUCKET in OFSA_FCAST_IRC_STRCT_CHG_BKT exist in OFSA_FCAST_IRC_STRCT_CHG_VAL. If any do not exist, insert the missing buckets and set the rate_change to zero.

Target Column OFSA_FCAST_RATES_BUCKETS	Source Column RATES_FORECAST	Special Conversion Logic
FCAST_RATES_SYS_ID	RATES_SYS_ID	See below
BUCKET_NUM	BUCKET	See below
BUCKET_TERM	N/A	plug 1
BUCKET_TERM_MULT	N/A	plug *M*

The logic to create the bucket data is as follows:

1. From the table RATES_FORECAST, select the RATES_SYS_ID and MAX (BUCKET).
2. Insert into the OFSA_FCAST_RATES_BUCKET a set of rows where BUCKET_NUM = 1 to the MAX (BUCKET) from step 1. For each row, automatically set the term and multiplier equal to 1 and *M* respectively.

Historical Rates ID

The Historical Rates ID conversion routine creates a **rate_manager_conv.log** file for messages and information about the conversion.

Specifying Historical Rates ID Priority

OFSA 3.5 and 4.0 allowed multiple Historical Rates IDs. However, Rate Manager enforces uniqueness for each IRC. Therefore, the Historical Rates conversion routine renames any duplicate Interest Rate Codes or Names during the conversion. Use the OFSA_TEMP_IRC_45 table to specify which Historical Rates IDs to convert and the precedence for converting them in the event that any duplicates exist. This table is also used for specifying the base currency for each Historical Rate ID to be converted.

If you designate Historical Rates IDs in OFSA_TEMP_IRC_45, only the IDs specified in this table are converted. Any other Historical Rates IDs that exist in the database but are not specified in OFSA_TEMP_IRC_45 are not converted.

If you do not designate any Historical Rates IDs in OFSA_TEMP_IRC_45, then the conversion routine processes the IDs in the ascending order of the rates_sys_id in IDT_RATES.

OFSA_TEMP_IRC_45

Column	Description
RATES_SYS_ID	Unique identifier for the Historical Rates ID (as defined in RATES_HISTORY table). This is the primary key of this table.
PRIORITY	Priority of conversion is a positive integer starting with 1. This column value has to be unique.
BASE_CURRENCY_CD	The base currency for the IRC. This is a 3 digit ISO currency code. It must be a valid ISO Currency Code from the list of Functional Currencies in Chapter 12, "FDM Database Upgrade Process".

If the Historical Rates ID has priority 1, the associated Historical Rates ID have the IRC numbering and name preserved in Rate Manager. All other priorities greater than 1 have the IRC numbering and name preserved if they are unique.

The conversion routine records the mapping of the old Interest Rate Codes and Name to the new Interest Rate Codes and Names in a temporary table named OFSA_TEMP_IRC_MAPPING_45. The temporary table has the following format:

OFSA_IRC_TEMP_MAPPING

Column	Description
SYS_ID_NUM	Unique identifier for the Historical Rates ID (as defined in RATES_HISTORY table)

Column	Description
OLD_INTEREST_RATE_CD	Old interest rate code from IDT_RATES table.
NEW_INTEREST_RATE_CD	Newly assigned value for interest rate code. This column is also the unique key.
OLD_IRC_NAME	Old description (IRC name) from IDT_RATES
NEW_IRC_NAME	Newly assigned IRC Name. The description column in IDT_RATES is 80 characters long. The new IRC name will be truncated to 60 characters.
BASE_CURRENCY_CD	The base currency for the IRC. This is a 3 digit ISO currency code. It must be one of the seeded currency codes in OFSA_CURRENCIES

When the conversion routine processes the priority 1 Historical Rates ID, it preserves the IRC number. Then, it processes the priority 2 ID. If the IRCs in this ID are unique, the conversion routine preserves the numbering. However, if it encounters a duplicate IRC, it renumbers it to 1001. The second time it encounters another duplicate IRC, it renumbers to 1002, then 1003, 1004, and so on.

When the conversion routine processes the priority 1 Historical Rates ID, it preserve the IRC name. Then, it processes the priority 2 ID. If the IRC name in this ID are unique, the conversion routine preserves the name. However, if it encounters a duplicate IRC, it renames it to IRC| |<interest_rate_cd>.

The temporary tables are renamed to O_XXXX and they are available to the user after the upgrade process is complete. Refer to O_OFSA_TEMP_IRC_MAPPING_45 to determine how the conversion routine reassigned interest rate code values.

Target Column OFSA_IRCS	Source Column TEMP_IRC_MAPPING_45	Special Conversion Logic
INTEREST_RATE_CD	INTEREST_RATE_CD	Reassign code values for overlapping IRCs.

Target Column OFSA_IRCS	Source Column TEMP_IRC_MAPPING_45	Special Conversion Logic
IRC_NAME	IRC_NAME	Reassign name for overlapping IRC name.
IRC_FORMAT_CD	0	Apply the value 0 (Zero Coupon Yield) in all cases.
ISO_CURRENCY_CD	BASE_CURRENCY_CD	
COMPOUND_BASIS_CD	150	Apply the value 150 (Annual) in all cases.
ACCRUAL_BASIS_CD	3	Apply the value 3 (Actual / Actual) in all cases.

If any Interest Rate Codes or Names were reassigned because of the existence of duplicates, you may need to update other database information for the reassigned values. Interest Rate Codes are used in the following tables and / or processes:

1. All instrument tables
2. Forecast Rates ID within Risk Manager
3. Rate Index ID within Risk Manager
4. Prepayment ID within Risk Manager and Transfer Pricing
5. Transfer Pricing ID within Transfer Pricing
6. Forecast Balance ID within Risk Manager
7. Leaf Characteristics ID within Risk Manager
8. Transaction Strategy ID within Risk Manager
9. Formula Leaves ID within Risk Manager
10. Discount Rates ID within Risk Manager
11. Stochastic Process ID (Valuation Curve Code) within Risk Manager
12. Data Filter Ids within all modules
13. Formula Ids within all modules
14. Report Ids within all modules
15. Repricing Patterns within Risk Manager and Transfer Pricing
16. Budgeting & Planning references

Within the Rate Manager module, you may want to consolidate date and rate information from interest rate codes that had existed across Historical Rates IDs. For example, you may have maintained only the last years' data in one Historical Rates ID and all historical dates prior to that for the same Interest Rate Code in another Historical Rates ID. In this case, you must consolidate the interest rate data within Rate Manager by cut and paste of rate data from one IRC into another IRC.

Interest Rate Terms

The terms for each IRC are recorded in the OFSA_IRC_RATE_TERMS table. This table stores all terms associated with each IRC. A summary of the data movement is provided:

Target Column OFSA_IRC_RATE_TERMS	Source Column TEMP_IRC_MAPPING_45	Special Conversion Logic
INTEREST_RATE_CD	INTEREST_RATE_CD	Use reassigned code values.
INTEREST_RATE_TERM	INTEREST_RATE_TERM	
INTEREST_RATE_TERM_MULT	INTERERST_RATE_ TERM_MULT	Must be D, M, Y.

For each interest rate code converted to the tables, a series of terms are also populated. The logic for converting this data is as follows:

1. Join the TEMP_IRC_MAPPING_45 table to the RATES_HISTORY table by matching on the IRC and Rates Sys ID where the origination date is equal to 01/01/1901. This isolates the rows from the RATES_HISTORY table that contain term point data.
2. For each row in the joined table, insert a row into the OFSA_IRC_RATE_TERMS table, as follows:
   - OFSA_IRC_RATE_TERMS. INTEREST_RATE_CD = Joined table.NEW_ INTEREST_RATE_CD
   - OFSA_IRC_RATE_TERMS. Interest_rate_term = RATES_HISTORY.Interest_ rate_term
   - OFSA_IRC_RATE_TERMS. Interest_rate_term_mult = RATES_ HISTORY.Interest_rate_mult

3. Validate that all terms represented in the OFSA_IRC_RATE_HIST exist in the OFSA_IRC_RATE_TERMS table. If any terms exist in the history table that do not exist in the term table, add the term to the term table. This validation is performed by comparing distinct values for INTEREST_RATE_CD, INTEREST_RATE_TERM, and INTEREST_RATE_TERM_MULT from OFSA_IRC_RATE_HIST table.

Rates Conversion

Target Column OFSA_IRC_RATE_HIST	Source Column RATES_HISTORY	Special Conversion Logic
EFFFECTIVE_DATE	ORIGINATION_DATE	Ignore when row date is 01/01/1901
INTEREST_RATE_CD	see Special Conversion Logic.	Use reassigned code values from TEMP_IRC_MAPPING_45.INTEREST_RATE_CD.
INTEREST_RATE_TERM	INTEREST_RATE_TERM	Ignore row when interest rate term is –1.
INTEREST_RATE_TERM_MULT	INTEREST_RATE_MULT	Must be D, M, Y.
INTEREST_RATE	INTEREST_RATE	
RATE_DATA_SOURCE_CD	1	Apply 1 (User Input) in all cases.
LAST_MODIFIED_DATE	Current Date	Use date when conversion routine is processed.

Records in RATES_HISTORY table with the following attributes are not converted.

- Interest_rate_term = -1 sets the date up as available date for data entry, it usually has interest rate of 0
- Origination_date = 01/01/1901 sets the IRC for term structure setup

The conversion logic is as follows:

1. Join the OFSA_TEMP_IRC_MAPPING_45 table to the RATES_HISTORY table by matching on the IRC and Rates Sys ID where the origination date is not equal to 01/01/1901 or the interest rate term is not equal to -1.

2. Validate values, as follows:
   - If the interest rate term is <=0, ignore the record and log it as an error message in rate_manager_conv.log in the log directory.
   - If the interest rate mult is not equal to D, M, or Y, then ignore the record and log it as an error message in rate_manager_conv.log in the log directory.
3. Then determine which table to write the rate into.
   - OFSA_IRC_RATE_HIST. INTEREST_RATE_CD = Joined table.NEW_ INTEREST_RATE_CD
   - OFSA_IRC_RATE_HIST. Effective_date = RATES_HISTORY.Origination_ date
   - OFSA_IRC_RATE_HIST.Data_source_cd = 1 (all converted rate is coded as User Input)
   - OFSA_IRC_RATE_HIST.Interest_rate = RATES_HISTORY.Interest_rate
   - OFSA_IRC_RATE_HIST.Last_modified_date is automatically populated with system date upon insertion
   - OFSA_IRC_RATE_HIST. Interest_rate_term = RATES_HISTORY.Interest_ rate_term
   - OFSA_IRC_RATE_HIST. Interest_rate_term_mult = RATES_ HISTORY.Interest_rate_mult

Leaf Characteristics ID

The Leaf Characteristics ID conversion routine creates a **leaf_charc_conv.log** file for messages and information about the conversion.

The mapping logic to the new FDM 4.5 data structure is as follows:

Target Column OFSA_IDT_LEAF_CHARACTERISTICS	Source Column OFSA_IDT_TM_DETAILS	Special Conversion Logic
LEAF_CHARAC_SYS_ID	SYS_ID_NUM	Must exist in OFSA_ CATALOG_OF_IDS where ID_TYPE = 309
LEAF_NODE	LEAF_NODE	
ISO_CURRENCY_CD	N / A	000

Target Column OFSA_IDT_LEAF_CHARACTERISTICS	Source Column OFSA_IDT_TM_DETAILS	Special Conversion Logic
FIELD_NUM	FIELD_NUM	
INT_VAL	INT_VAL	
FLOAT_VAL	FLOAT_VAL	
DATE_VAL	DATE_VAL	
DEC_VAL	DEC_VAL	
N/A	LEAF_NUM_ID	Same as Leaf_Num_ID stored in Catalog Of IDs; no longer needed in Leaf Characteristics table.

For every distinct combination of valid Leaf_Charac_Sys_ID and Leaf_Node in OFSA_IDT_LEAF_CHARACTERISTICS, the conversion routine creates two additional rows:

Target Column OFSA_IDT_LEAF_CHARACTERISTICS	Special Conversion Logic
LEAF_CHARAC_SYS_ID	selected from distinct Leaf_ Charac_Sys_ID + Leaf_Node
LEAF_NODE	selected from distinct Leaf_ Charac_Sys_ID + Leaf_Node
ISO_CURRENCY_CD	000 (Special Default Currency)
FIELD_NUM	1st inserted row: Plug: 5 (Currency Gain/Loss Basis) 2nd inserted row: Plug: 6 (Pay Equivalent Compounding Convention)
INT_VAL	For new Field_Num 5: Plug: 0 (means *Temporal*)For new Field_Num 6: Plug: 0 (means switch is off)
FLOAT_VAL	0

Target Column OFSA_IDT_LEAF_CHARACTERISTICS	Special Conversion Logic
DATE_VAL	'01/01/1960'
DEC_VAL	0

Maturity Strategy ID

1. OFSA_IDT_MATURITY and OFSA_MATURITY_AUXILIARY
   - Update all the values in ISO_CURRENCY_CD column to 000 (Special Default Currency)

Pricing Margin ID

1. OFSA_IDT_PRICING_MARGIN
   - Update all the values in ISO_CURRENCY_CD column to 000.

RM Process ID

If you are upgrading from OFSA 3.5, the FDM Database Upgrade Process executes the logic described in the Upgrades from OFSA 3.5 section before executing the logic described in the All Upgrades section.

Upgrades from OFSA 3.5

Process Type

Process type is a new input for the Risk Manager Process ID. The following two process types are available:

- Scenario-based Processing
- Stochastic Processing

Any Budget and Planning processes are converted to Scenario-based IDs.

Output Options

Output options include organizational unit output, audit trail output and financial element output.

For Process IDs that were formerly Budget and Planning processes, one optional financial element set is selected, during the conversion, for repricing financial elements.

New Business Leaves

New Business Leaves are no longer necessary for the Process ID and are not converted.

All Upgrades

1. OFSA_IDT_TM_PROCESS
   - Update all the values in REPORTING_CURRENCY_CD column to be same as OFSA_DB_INFO.FUNCTIONAL_CURRENCY_CD
2. OFSA_TM_SCENARIO_ASSUMP
   - OUTPUT_BY_ORG_FLG is the new name for what used to be TWO_LEAF_FLG
   - Update all the values in OUTPUT_BY_CURRENCY_FLG column to 0
   - Update all the values in OUTPUT_FCAST_RATES_FLG column to 0
   - Update all the values in CONSOLIDATED_OUTPUT_FLG column to 0.
3. OFSA_TM_STOCH_ASSUMP
   - For each Term Structure ID map TS_MODEL_CD and SMOOTHING_METHD_CD to all the RM Process IDs in OFSA_TM_STOCH_ASSUMP tables that use that Term Structure ID. Term Structure ID is now an obsolete ID.

The Term Structure ID mapping is as follows:

Target Column OFSA_TM_STOCH_ASSUMP	Source Column IDT_TERM_STRUCTURE	Special Conversion Logic
TM_PROCESS_SYS_ID		
TS_MODEL_CD	TS_MODEL_CD	
SMOOTHING_METHOD_CD	SMOOTHING_METHD_CD	

The rest of the columns in OFSA_TM_STOCH_ASSUMP remain the same.

Term Structure ID

The FDM upgrade process converts Term Structure IDs to RM Process IDs. Term Structure IDs no longer exist in Risk Manager version 4.5. Refer to the RM Process ID conversion logic for more information.

TP Process ID

The FDM upgrade process migrates data from the IDT_PROCESS table to three new tables for version 4.5. These new tables are:

- OFSA_IDT_PROCESS
- OFSA_TP_PROC_TABLES
- OFSA_TP_RATE_PROPAGATIONS

The conversion mappings are as follows:

OFSA_IDT_PROCESS

Target Column OFSA_IDT_PROCESS	Source Column IDT_PROCESS	Special Conversion Logic
TP_PROCESS_SYS_ID	SYS_ID_NUM	Must exist in OFSA_ CATALOG_OF_IDS where id_type=204.
CLIENT_SERVER_FLG	SWITCH_STATE	See below
CALC_MODE_CD	SWITCH_STATE	See below
TRANSFER_PRICE_SYS_ID	PROCESS_SYS_ID	See below
PREPAY_SYS_ID	PROCESS_SYS_ID	See below
FILTER_TYPE	ID_TYPE	See below
FILTER_SYS_ID	PROCESS_SYS_ID	See below
DTL_CASHFLOW_FLG	SWITCH_STATE	See below
SKIP_NONZERO_TRANS_ RATE_FLG	SWITCH_STATE	See below
SKIP_NONZERO_OPT_COST_ FLG	NULL	None

Target Column OFSA_IDT_PROCESS	Source Column IDT_PROCESS	Special Conversion Logic
TRANS_RATE_PROPAGATE_ FLG	SWITCH_STATE	See below
TRANS_RATE_CALC_FLG	SWITCH_STATE	See below
TRANS_RATE_MIGRATE_FLG	SWITCH_CODE	See below
OPTION_COST_PROPAGATE_ FLG	NULL	None
OPTION_COST_CALC_FLG	NULL	None
OPTION_COST_MIGRATE_FLG	NULL	None

All flags and codes column logic

IF IDT_PROCESS.PROCESS_TARGET = 2:

1. CLIENT_SERVER_FLG => IDT_PROCESS.SWITCH_CODE = 9
   a. IDT_PROCESS.SWITCH_STATE = 1 then CLIENT_SERVER_FLG = 0 else CLIENT_SERVER_FLG = 1.
   b. Default: 1 (Use client)
2. CALC_MODE_CD => IDT_PROCESS.SWITCH_CODE = 3
   a. CALC_MODE_CD = IDT_PROCESS.SWITCH_STATE
   b. Default: 0 (Standard mode)
3. DTL_CASHFLOW_FLG =>IDT_PROCESS.SWITCH_CODE =17
   a. DTL_CASHFLOW_FLG = IDT_PROCESS.SWITCH_STATE
   b. Default: NULL (Write Detail Cash Flow is not checked)
4. SKIP_NONZERO_TRANS_RATE_FLG => IDT_PROCESS.SWITCH_CODE =19
   a. SKIP_NONZERO_TRANS_RATE_FLG=IDT_PROCESS.SWITCH_STATE
   b. Default: NULL (Skip Non Zero Transfer Rates is not checked)
5. TRANS_RATE_PROPAGATE_FLG => IDT_PROCESS.SWITCH_CODE =18
   a. TRANS_RATE_PROPAGATE_FLG = IDT_PROCESS.SWITCH_STATE
   b. Default: NULL (Transfer Rate Propagate is not checked)
6. TRANS_RATE_CALC_FLG => IDT_PROCESS.SWITCH_CODE =18

a. If IDT_PROCESS.SWITCH_CODE = 18 has an entry in IDT_PROCESS then TRANS_RATE_CALC_FLG = 0 else TRANS_RATE_CALC_FLG = 1.

b. Default: 1 (Transfer Rate Calculation Flag is checked)

7. TRANS_RATE_MIGRATE_FLG

   a. IF IDT_PROCESS.SWITCH_CODE = 0 (Instrument table is checked) and IDT_PROCESS.SWITCH_CODE = 1 (Ledger_stat table is checked) have entries for this TP_PROCESS_SYS_ID then TRANS_RATE_MIGRATE_FLG = 1 else TRANS_RATE_MIGRATE_FLG=0.

   b. Default: 0 (Transfer Rate Migrate is not checked)

Note: If the SWITCH_CODE does not have entries for this TP_process_sys_id, then the conversion routine uses the default values.

All Assumption IDs column logic

IF IDT_PROCESS.PROCESS_TARGET = 1:

1. TRANSFER_PRICE_SYS_ID => IDT_PROCESS.ID_TYPE=200

   a. IDT_TP_PROCESS.TRANSFER_PRICE_SYS_ID = IDT_PROCESS.PROCESS_SYS_ID

   b. IDT_TP_PROCESS.PROCESS_SYS_ID has to exist in OFSA_CATALOG_OF_IDS where id_type=200.

   c. Default: 0

2. FILTER_SYS_ID and FILTER_TYPE => IDT_PROCESS.ID_TYPE = 4,8,21(Data, Tree and Group Filter)

   a. IDT_PROCESS.FILTER_TYPE = IDT_PROCESS.ID_TYPE

   b. IDT_PROCESS.FILTER_SYS_ID = IDT_PROCESS.PROCESS_SYS_ID

   c. IDT_PROCESS.PROCESS_SYS_ID has to exist in OFSA_CATALOG_OF_IDS where id_type = IDT_PROCESS.ID_TYPE. If not exist, do not migrate it over.

   d. Default: NULL

3. PREPAY_SYS_ID = > IDT_PROCESS.ID_TYPE=300

   a. IDT_PROCESS.FILTER_SYS_ID = IDT_PROCESS.PROCESS_SYS_ID

b. IDT_PROCESS.PROCESS_SYS_ID has to exist in OFSA_CATALOG_OF_IDS where id_type = 300. If not exist, do not migrate it over.

c. Default: NULL

Note: If the ID_TYPE does not have entries for this TP_PROCESS_SYS_ID, then the conversion routine uses the default values.

OFSA_TP_PROC_TABLES

Target Column OFSA_TP_PROC_TABLES	Source Column IDT_PROCESS	Special Conversion Logic
TP_PROCESS_SYS_ID	SYS_ID_NUM	Must exist in OFSA_IDT_TP_PROCESS.
TABLE_NAME	TABLE_NAME	If the process id has IDT_PROCESS.PROCESS_TARGET = 2 and IDT_PROCESS.SWITCH_CODE = 1, add 'LEDGER_STAT' as the process target table. For each entry that has IDT_PROCESS.PROCESS_TARGET=3, add IDT_PROCESS.TABLE_NAME as the process target table.

OFSA_TP_RATE_PROPAGATIONS

Target Column OFSA_TP_RATE_PROPAGATIONS	Source Column IDT_PROCESS	Special Conversion Logic
TP_PROCESS_SYS_ID	SYS_ID_NUM	Must exist in OFSA_IDT_TP_PROCESS.
PROPAGATE_TARGET_TABLE	TABLE_NAME	For each TABLE_NAME in OFSA_TP_PROC_TABLES except LEDGER_STAT.

Target Column OFSA_TP_RATE_PROPAGATIONS	Source Column IDT_PROCESS	Special Conversion Logic
PROPAGATE_SOURCE_TABLE	TABLE_NAME	Where IDT_ PROCESS.PROCESS_ TARGET = 5 and IDT_ PROCESS.PROCESS_SYS_ ID = PROCESS_SYS_ID of the table_name used for the PROPAGATE_TARGET_ TABLE field.
PROPAGATE_LAG_TERM	SWITCH_STATE	Where IDT_ PROCESS.PROCESS_ TARGET = 5 and IDT_ PROCESS.PROCESS_SYS_ ID = PROCESS_SYS_ID of the table_name used for the PROPAGATE_TARGET_ TABLE field.
PROPOGATE_LAG_MULT	SWITCH_CODE	Where IDT_ PROCESS.PROCESS_ TARGET = 5 and IDT_ PROCESS.PROCESS_SYS_ ID = PROCESS_SYS_ID of the table_name used for the PROPAGATE_TARGET_ TABLE field. IDT_PROCESS.SWITCH_ CODE => 1 = ‘M’, 2 = ‘Y’, 3=’D’.

Transaction Strategy ID

The Transaction Strategy ID conversion routine creates a **trans_strategy_conv.log** file for messages and information about the conversion.

The mapping logic to the new FDM 4.5 data structure is as follows:

Target Column OFSA_IDT_TRANS_STRATEGIES	Source Column OFSA_IDT_TM_DETAILS	Special Conversion Logic
TRANSACTION_SYS_ID	SYS_ID_NUM	Must exist in OFSA_ CATALOG_OF_IDS where ID_TYPE = 306

Target Column OFSA_IDT_TRANS_STRATEGIES	Source Column OFSA_IDT_TM_DETAILS	Special Conversion Logic
TRANSACTION_NUM	LEAF_NUM_ID	
LEAF_NODE	LEAF_NODE	
FIELD_NUM	FIELD_NUM	Add 1 to Field_Num where existing Field_Num > 0. This allows insertion of row for Currency Code.
ISO_CURRENCY_CD	N / A	Plug with OFSA_DB_ INFO.Functional_Currency_ Cd.
INT_VAL	INT_VAL	
FLOAT_VAL	FLOAT_VAL	
DATE_VAL	DATE_VAL	
DEC_VAL	DEC_VAL	

For every distinct combination of valid Transaction_Sys_ID and Transaction_Num in OFSA_IDT_TRANS_STRATEGIES, the conversion routine creates an additional row for the ISO_CURRENCY_CD:

Target Column OFSA_IDT_TRANS_STRATEGIES	Special Conversion Logic
TRANSACTION_SYS_ID	select from distinct Transaction_Sys_ID + Transaction_Num
TRANSACTION_NUM	select from distinct Transaction_Sys_ID + Transaction_Num
LEAF_NODE	associated Leaf_Node
FIELD_NUM	1
ISO_CURRENCY_CD	OFSA_DB_INFO. FUNCTIONAL_ CURRENCY_CD
INT_VAL	0

Target Column OFSA_IDT_TRANS_STRATEGIES	Special Conversion Logic
FLOAT_VAL	0
DATE_VAL	'01/01/1960'
DEC_VAL	0

Transfer Pricing ID

The FDM upgrade process migrates data from the IDT_TRANSFER_PRICE and TP_AUXILIARY tables to 3 new tables for version 4.5. These new tables are:

- OFSA_IDT_TRANSFER_PRICE
- OFSA_TP_REDEMPTION_CURVE_DTL
- OFSA_TP_UNPRICED_ACCT_DTL

The conversion mappings are as follows:

OFSA_IDT_TRANSFER_PRICE

Target Column OFSA_IDT_TRANSFER_PRICE	Source Column IDT_TRANSFER_PRICE	Special Conversion Logic
TRANSFER_PRICE_SYS_ID	SYS_ID_NUM	Must exist in OFSA_CATALOG_OF_IDS where id_type=200.
LEAF_VALUE	LEAF_NODE	NONE
LEAF_DATA_SOURCE_CD	METHOD_TYPE	If IDT_TRANSFER_PRICE.METHOD_TYPE= 7 then LEAF_DATA_SOURCE_CD = 2 else LEAF_DATA_SOURCE_CD = 1.

Target Column OFSA_IDT_TRANSFER_PRICE	Source Column IDT_TRANSFER_PRICE	Special Conversion Logic
TP_CALC_METHOD_CD	METHOD_TYPE	IDT_TRANSFER_ PRICE.METHOD_TYPE = 9(Migrate Rates Only method) is no longer supported in release 4.5. All of the records that have method_type=9 will have TP_CALC_METHOD_CD set to 0 (None).
GROSS_RATE_FLG	0	Apply to all. The Model With Gross Rates switch moves from TP Process ID to Transfer Pricing ID.
INTEREST_RATE_CD	INTEREST_RATE_CD	NONE
YIELD_CURVE_TERM	MATURITY_FREQ	NONE
YIELD_CURVE_MULT	MATURITY_FREQ_ MULT	NONE
HISTORICAL_TERM	HISTORY_FREQ	NONE
HISTORICAL_MULT	HISTORY_FREQ_MULT	NONE
ASSIGNMENT_DATE_CD	ORG_DATE_CD	If IDT_TRANSFER_ PRICE.ORG_DATE_CODE= 0 then ASSIGNMENT_ DATE_CD = NULL else ASSIGNMENT_DATE_CD = IDT_TRANSFER_ PRICE.ORG_DATE_CODE.
OPTION_COST_METHOD_CD	0	Apply to all
TARGET_BAL_CD	NULL	NONE
RATE_SPREAD	RATE_SPREAD	NONE
LAG_TERM	LAG_FREQ	NONE
LAG_MULT	LAG_FREQ_MULT	NONE
ACROSS_ORG_UNIT_FLG	ACROSS_ORG_UNIT	NONE

Target Column OFSA_IDT_TRANSFER_PRICE	Source Column IDT_TRANSFER_PRICE	Special Conversion Logic
MID_PERIOD_REPRICE_FLG	MID_PERIOD	If IDT_TRANSFER_ PRICE.MID_PERIOD > 0 then MID_PERIOD_ REPRICE_FLG = 1 else MID_PERIOD_REPRICE_ FLG = 0

Data is not converted where:

1. IDT_TRANSFER_PRICE.SYS_ID_NUM does not exist in OFSA_CATALOG_ OF_IDS.
2. IDT_TRANSFER_PRICE.INTEREST_RATE_CD does not exist in OFSA_IRCS.

OFSA_TP_REDEMPTION_CURVE_DTL

If IDT_TRANSFER_PRICE.METHOD_TYPE = 8, convert the Transfer Pricing ID data from TP_AUXILIARY to OFSA_TP_REDEMPTION_CURVE_DTL using the following mappings:

Target Column OFSA_TP_REDEMPTION_CURVE_DTL	Source Column TP_AUXILIARY	Special Conversion Logic
TRANSFER_PRICE_SYS_ID	TRANSFER_SYS_ID	Must exist in OFSA_IDT_ TRANSFER_PRICE
LEAF_VALUE	LEAF_NODE	Must exist in OFSA_IDT_ TRANSFER_PRICE
INTEREST_RATE_CD	See Special Conversion Logic	Retrieve from IDT_ TRANSFER_ PRICE.INTEREST_RATE_ CD WHERE TP_ AUXILIARY.TRANSFER_ SYS_ID = IDT_TRANSFER_ PRICE.SYS_ID_NUM and TP_AUXILIARY.LEAF_ NODE=IDT_TRANSFER_ PRICE.LEAF_NODE.
INTEREST_RATE_TERM	MATURITY_FREQ	NONE
INTEREST_RATE_TERM_MULT	MATURITY_FREQ_ MULT	NONE

Target Column OFSA_TP_REDEMPTION_CURVE_DTL	Source Column TP_AUXILIARY	Special Conversion Logic
PERCENTAGE	PERCENTAGE	NONE

Data is not converted where:

1. TP_AUXILIARY.TRANSFER_SYS_ID + TP_AUXILIARY.LEAF_NODE does not exist in OFSA_IDT_TRANSFER_PRICE.
2. Combination of INTEREST_RATE_CD,INTEREST_RATE_TERM,INTEREST_RATE_TERM_MULT does not exist in OFSA_IRC_RATE_TERMS.

OFSA_TP_UNPRICED_ACCT_DTL

If IDT_TRANSFER_PRICE.METHOD_TYPE = 7, convert the Transfer Pricing ID data from TP_AUXILIARY to OFSA_TP_UNPRICED_ACCT_DTL using the following mappings:.

Target Column OFSA_TP_UNPRICED_ACCT_DTL	Source Column TP_AUXILIARY	Special Conversion Logic
TRANSFER_PRICE_SYS_ID	TRANSFER_SYS_ID	Must exist in OFSA_IDT_TRANSFER_PRICE
LEAF_VALUE	LEAF_NODE	Must exist in OFSA_IDT_TRANSFER_PRICE
SOURCE_LEAF_VALUE	TRANSFER_LEAF	NONE

Data is not converted where:

1. The combination of TP_AUXILIARY.TRANSFER_SYS_ID and TP_AUXILIARY.LEAF_NODE does not exist in OFSA_IDT_TRANSFER_PRICE.

12

FDM Database Upgrade Process

This chapter discusses the procedure for upgrading Oracle Financial Services (OFSA) 3.5/4.0 version databases to the Oracle Financial Data Manager (FDM) database version 4.5. This procedure supports upgrading from OFSA Release 3.5 and 4.0 databases.

The following topics are covered in this chapter:

- Overview of the 4.5 Upgrade Process
- Required Oracle Parameters for the FDM Upgrade Process
- Running the Metadata Migration
- Running the Upgrade Procedure
- Password Encryption Changes
- OFSA Database Problem Conditions and Solutions

The upgrade scripts for this procedure are on the same media as the server-centric software. The default location for both the scripts and working directory is the OFSA_INSTALL/dbs/<OFSA release> subdirectory of your OFSA installation

directory. In this chapter, OFSA_INSTALL is the convention used to indicate where the OFSA software is installed in your directory structure.

Note: Oracle recommends that you run the Migrate_Check step of the upgrade procedure at least 3 weeks prior to beginning the FDM Database Upgrade Process. The Migrate_Check step identifies data inconsistencies with your database that must be resolved in order for the FDM upgrade to proceed. In order to resolve these issues, you need assistance from the OFSA end-users. Oracle recommends that you reserve sufficient time to resolve these issues prior to continuing with the FDM database upgrade process. The Migrate_Check step does not perform any structural changes to your database, so you can run the Migrate_Check step as many times as you need while continuing to use your database with OFSA 3.5 or 4.0.

Note: Market Manager version 4.0.3 is compatible with an FDM 4.5 database with the Market Manager objects installed. The FDM 4.5 database upgrade process therefore preserves the database objects required for Market Manager version 4.0.3.

Caution:

- The scripts must remain in a single directory during the upgrade procedure.
- The following tablespaces are required for the database upgrade procedure: DATA_TS and INDEX_TS. If these are not available, you cannot complete the database upgrade.

Before running the upgrade procedure, be sure to review Chapter 11, "Upgrading from OFSA 3.5/4.0" for database changes in FDM 4.5.

Overview of the 4.5 Upgrade Process

The 4.5 Database Upgrade Process introduces significant changes to the FDM database structure. The changes include:

- Elimination of Password Encryption for individual users
- OFSA_ prefix for all application database objects
- Addition of database structures to support Multiple languages
- Addition of database structures to support Multiple currencies
- Creation of individual Code Description tables

These changes are detailed in Chapter 11, "Upgrading from OFSA 3.5/4.0". It is recommended that you review this chapter before beginning the upgrade process.

The upgrade process consists of running the following steps in order:

- Running the Metadata Migration
- Running the Upgrade procedure
- Installing and Configuring Discoverer (See Chapter 13, "Installing and Configuring Discoverer".)

Limitations to the Database Upgrade Process

Before you begin, review the following limitations to assess the impact on your upgrade:

Instrument Table Indexes

The upgrade process cannot create indexes on your instrument tables intelligently because each organization customizes its database according to its unique needs. Therefore, no instrument or client data table indexes are created as part of the upgrade. It is left to you to create these indexes according to the general guidelines discussed in this manual.

Multiple LEDGER_STAT Tables (data_code = 7)

FDM 4.5 only supports a single LEDGER_STAT table. The FDM Database Upgrade Process does not migrate any other table names in SYSTEM_INFO assigned to data_code=7 (the Ledger_Stat data_code). Only the LEDGER_STAT table is migrated. OFSA versions 3.5 and 4.0 also did not support having table names other than LEDGER_STAT assigned to data_code 7.

If you do have table names other than LEDGER_STAT assigned to data_code=7, assign them to data_code = 10 (User Defined) prior to beginning the FDM upgrade process. The FDM Database Upgrade Process then migrates those objects as User Defined to the new FDM Metadata structure.

Seeded Data Tables and Ranges Affected by the Upgrade

Oracle defines seeded data as: 1) Data placed in the database to run the OFS applications properly; and 2) Data that makes up the IDs shipped with the database.

For some tables, Oracle reserves the entire table for internal use. This means that any user-defined rows in the tables are deleted by the upgrade process. For other tables, Oracle reserves only a seeded range. User-defined rows are permitted within a specified range. For a complete list of seeded data tables and ranges, refer to Chapter 16, "FDM Object Management".

Required Oracle Parameters for the FDM Upgrade Process

The FDM Upgrade Process requires that you set the following initialization parameters prior to running the upgrade. Specify these parameters in the init<dbname>.ora file where <dbname> is the Oracle SID of your database. After specifying these parameters, you must shutdown your database and restart it prior to running the FDM Upgrade Process.

For a list of Oracle parameters recommended for your FDM database, refer to Chapter 10, "FDM Database Installation".

> **Caution:** The initialization parameters specified in this section are required for the FDM database. If you do not set your Oracle initialization parameters as specified, your upgrade process may not complete successfully and your FDM database may not function properly.

- **compatible**

  FDM requires that this parameter is set to 8.1.6 or higher.

  COMPATIBLE lets you use a new release, while at the same time guaranteeing backward compatibility with an earlier release. This ability is helpful in case it becomes necessary to revert to the earlier release.

- **dml_locks**

  FDM requires that this parameter is set to at least 200.

  A DML lock is a lock obtained on a table that is undergoing a DML operation (insert, update, delete). DML_LOCKS specifies the maximum number of DML locks--one for each table modified in a transaction. The value should equal the grand total of locks on tables currently referenced by all users. For example, if three users are modifying data in one table, then three entries would be required. If three users are modifying data in two tables, then six entries would be required.

- **job_queue_processes**

  FDM requires that this parameter is set to at least 1 (maximum value is 36).

  JOB_QUEUE_PROCESSES specifies the number of SNPn job queue processes per instance (SNP0, ... SNP9, SNPA, ... SNPZ). Job queue processes process requests created by DBMS_JOB.

- **max_enabled_roles**

  FDM requires that the max_enabled_roles parameter is set to at least 60. This is because the FDM database creation and database upgrade processes create a number of seeded roles in the database instance.

  MAX_ENABLED_ROLES specifies the maximum number of database roles that users can enable, including roles contained within other roles.

- **open_cursors**

  Specifies the maximum number of open cursors (context areas) a session can have at once. This constrains a session from opening an excessive number of cursors. Oracle recommends a value of 500 to accommodate the FDM Database Upgrade Process and OFS applications multiprocessing.

- **shared_pool_size**

  SHARED_POOL_SIZE specifies in bytes the size of the shared pool. The shared pool contains shared cursors, stored procedures, control structures, and other structures. The FDM Database Upgrade Process requires an appropriate value for the shared_pool_size parameter in order to complete successfully. Oracle recommends a value of 50000000 or more for this parameter in order to ensure adequate resources for the FDM Database Upgrade Process.

Running the Metadata Migration

The OFSA 4.5 Database Upgrade Process includes a migration to a new metadata structure. This new metadata structure supports new features for object and security administration with the Financial Data Manager Administration application. Migration to this new metadata structure is the first step in the OFSA 4.5 Database Upgrade Process.

The Metadata Migration includes the following steps:

1. Review Migration Requirements
2. Prepare Database for Migration
3. Run the migrate_check.sql
4. Run the migrate.sql
5. Review Migrate Logs

Review Migration Requirements

Prior to running the Migration procedure, review the following conversion requirements:

Historical Rates Conversion

The Rate Manager application requires that the Historical Rates IDs from OFSA 3.5/4.0 are converted into a new format for version 4.5. Previously in OFSA 3.5/4.0, it was possible for Interest Rate Codes and Names to exist across different Historical Rates IDs. The Rate Manager application requires that Interest Rate Codes and Names are unique within the database. Because of this, Oracle recommends that you specify the precedence for Historical Rates ID for the conversion.

The Migrate procedure creates the OFSA_TEMP_IRC_45 during the initial execution. The Migrate procedure then prompts as follows:

```
The OFSA_TEMP_IRC_45 table is empty.  If you do not setup the
OFSA_TEMP_IRC_45 table, the DUP will convert your IRCs in the ascending
order of the Historical Rates ID sys_id_num.  It is recommended that
you setup the OFSA_TEMP_IRC_45 table before you run the migration process.

To setup this table exit the migration procedure.

Do you want to exit and setup the OFSA_TEMP_IRC_45 table?(y/n):
```

If you have multiple Historical Rates IDs in your OFSA 3.5/4.0 database that you want to convert to the new Rate Manager format, select *Y* to exit the Migrate procedure and setup your Historical Rates ID conversion information. If you actively use only one Historical Rates ID in OFSA 3.5/4.0, then you do not need to setup the OFSA_TEMP_IRC_45 table prior to running the Metadata Migration. In this case, select *N* to continue with the Migrate procedure.

Note: The first time you run the Migrate Check procedure, it creates an empty OFSA_TEMP_IRC_45 table. If you want to convert multiple Historical Rates ID, you should exit the Migrate procedure to setup your Historical Rates ID conversion precedence information and then launch Migrate a second time to complete the procedure.

Note: Refer to Chapter 11, "Upgrading from OFSA 3.5/4.0" for detailed information about the Historical Rates ID conversion.

Setup for OFSA_TEMP_IRC_45

The OFSA_TEMP_IRC_45 table is used to designate the Base Currency and conversion priority for the Historical Rates IDs. Use SQL*Plus insert statements to setup this table. For example:

```
insert into ofsa_temp_irc_45 (rate_sys_id, priority, base_currency_cd)
values (100332, 1, 'USD');
```

The RATE_SYS_ID value must be a valid Historical Rates ID. The PRIORITY value is the precedence in which want to convert your Historical Rates ID. Assign a value of 1 to your primary Historical Rates ID. The BASE_CURRENCY_CD is the ISO Currency used for the Historical Rates conversion. The For a list of valid Currencies, refer to Appendix A, "Functional Currencies".

If you have multiple Historical Rates IDs and you do not specify the Priority and Currency in the OFSA_TEMP_IRC_45 table, the upgrade process converts your IDs in the ascending order of the sys_id_num. The sys_id_num identifies each ID in the CATALOG_OF_IDS table. The conversion then renames and renumbers any cases where a duplicate IRC name exists across multiple Historical Rates IDS. Refer to O_OFSA_TEMP_IRC_MAPPING_45 to determine how the conversion routine reassigned interest rate code values.

Note: In most cases, only one Historical Rates ID needs to be converted into the 4.5 Rate Manager format. Only in situations where you are actively using multiple Historical Rates ID do you need to specify the conversion priority in the OFSA_TEMP_IRC_45 table. If you do designate Historical Rates Ids in the OFSA_TEMP_IRC_45, only the IDs specified in the table are converted. Any other Historical Rates Ids that exist in the database but are not specified in OFSA_TEMP_IRC_45 are not converted

Functional Currency

The Migration procedure requires the specification of a Functional Currency in the table OFSA_TEMP_DB_INFO. This table is created by the Migrate process during its initial execution. The first time that you run the Migrate procedure, this table is created (empty) and the Migrate procedure then records an error in the migrate_check.log file indicating that you need to specify the Functional Currency.

Note: The first time you run the Migrate Check procedure, it creates an empty OFSA_TEMP_DB_INFO table and reports an error that the Functional Currency is not specified in this table. Once you have specified a Functional Currency in this table, you need to launch Migrate Check a second time to verify that all errors are resolved before you proceed.

Functional Currency is defined as the currency of the primary economic environment in which an entity conducts its business. In a single currency environment, you need only specify your currency code. In a multiple currency environment, specify the currency used by the parent organization for financial statement reporting.

The FDM database upgrade process sets all Client Data Tables (such as Instrument tables, LEDGER_STAT, etc.) with the ISO_CURRENCY_CD column to default to the specified Functional Currency whenever a value is not explicitly designated during an insert.

If you do not specify a Functional Currency, the Migrate procedure outputs the following error to the **migrate_check.log** file:

```
ERROR! Functional Currency Code not defined in OFSA_TEMP_DB_INFO.
```

To resolve this you must update the OFSA_TEMP_DB_INFO table with a valid Currency from the list of Functional Currencies in Appendix A, "Functional Currencies". Refer to OFSA Database Problem Conditions and Solutions for details on this solution.

Prepare Database for Migration

Follow these steps to prepare the database.

1. Unset the ORACLE_PATH and SQL_PATH environment variables so that custom SQL scripts are not executed in place of the upgrade SQL scripts. The command line entries are:

   ```
   unset ORACLE_PATH
   unset SQL_PATH
   ```

   Caution: Unsetting these environment variables is essential. If you do not, you will create problems during the upgrade procedure.

2. Verify that your $ORACLE_SID points to the correct ORACLE instance on the server.

   ```
   echo $ORACLE_SID
   ```

3. Remove any tables in the current schema that have the prefix O_ (the letter *O* followed by an underscore).

   The following SQL statement identifies tables with the O_ prefix:

   ```
   SELECT distinct table_name
   FROM all_tables
   WHERE table_name like 'O\_%' escape '\';
   ```

   The O_ prefix is reserved for the OFSA database upgrade procedure. If you do not remove these tables from your system, you receive a problem warning message in the check.log file.

4. Ensure that the minimum following space is available before you begin:

SYSTEM tablespace	85 MB (minimum) if initjvm.sql not run
	10 MB (minimum) if initjvm.sql already run
DATA_TS tablespace	50 MB (minimum)
INDEX_TS tablespace	50 MB (minimum)
UNIX file system	10 MB for log files and temporary files

Note: These freespace requirements pertain only to completing the Migrate procedure. Additional freespace is required to complete the remaining steps of the FDM Database Upgrade Process. These additional freespace requirements are documented in Running the Upgrade Procedure.

5. Login to SQL*Plus as SYS and load the initjvm.sql package into the database.

   The initjvm package is required for FDM 4.5. This Oracle RDBMS package provides the necessary environment for java class to operate within the database. The package is normally found in the following directory location:

   $ORACLE_HOME/javavm/install/initjvm.sql

   To load the initjvm.sql package, go to the directory where the initjvm.sql script is located, login to SQL*Plus as the SYS user and type the following:

```
SQL> spool initjvm.log
SQL> @initjvm.sql
```

   Review the spooled logfile after the initjvm.sql script has completed in order to verify that there were no significant errors. The initjvm.sql script normally requires several minutes to complete.

   Note: Always spool to a logfile prior to running the initjvm.sql. After the script is complete, review the spooled logfile for errors. Ignore any object-not-found errors for drop table, drop index, drop package, and other drop object statements.

Note: It is critical to run the initjvm.sql script before proceeding with the update process. This package is required for the new FDM security framework. Be sure that you have at least 75MB of free space in the SYSTEM tablespace prior to running this package.

6. Shut down the Oracle listener.
7. Shut down and restart the database.

   This guarantees that no one but the System Administrator is accessing the database during the upgrade procedure and ensures that any new initialization parameters take effect.

Run migrate_check.sql

Follow these steps to run migrate_check.sql. In this section the OFSA_INSTALL/dbs/<OFSA release> directory is referred to as the database upgrade home directory, or <Upgrade home dir>.

Note: Oracle recommends that you run the Migrate_Check step of the upgrade procedure at least 3 weeks prior to beginning the FDM Database Upgrade Process. The Migrate_Check step identifies data inconsistencies with your database that must be resolved in order for the FDM upgrade to proceed. In order to resolve these issues you need assistance from the OFSA end-users. Oracle recommends that you reserve sufficient time to resolve these issues prior to continuing with the FDM database upgrade process. The Migrate_Check step does not perform any structural changes to your database, so you can run the Migrate_Check step as many times as you need while continuing to use your database with OFSA 3.5 or 4.0.

1. In UNIX, change the directory to the <Upgrade home dir> directory.
2. Log into SQL*Plus as SYS using the following command line entry:

   ```
   sqlplus SYS/<password>
   ```

3. Run the cr_owner.sql script, which is in the <Upgrade home dir> / master directory, to grant additional privileges to the database owner. It prompts you for the name of the OFSA database owner. Prior to running the script, use the SQL*Plus *spool* command to redirect any error messages to a log file.

   ```
   SQL> @master/cr_owner
   ```

   Ignore the following error messages when they appear:

   ```
   ORA-01920: user name <owner_name> conflicts with another user or role
   name
   ```

   ```
   ORA-01921: role name <name> conflicts with another user or role name
   ```

4. Connect as the database owner using the following command line entry:

   ```
   SQL> connect <owner_name>/<password>
   ```

5. Execute the pre-migrate check process using the following command line entry:

   ```
   SQL> @migrate/migrate_check <Password> <Upgrade home dir> <Sql*Loader
   executable>
   ```

   <Password> This is the password for the database owner, to invoke SQL*Loader.

   <Upgrade home dir> This is the full path to the database upgrade home directory as previously described. Do not enter a trailing / character.

   An example of a full path follows:

   ```
   /app/ofsa/ofsa4.0-092/dbs/450001010
   ```

   <Sql*Loader executable> This is the command used to execute SQL*Loader. For most Oracle installations, this command is sqlldr.

6. Review the migrate_check.log file created in the <Upgrade home dir> / log directory, for any problem conditions.

   If error conditions appear in the migrate_check.log file, refer to "OFSA Database Problem Conditions and Solutions" to resolve them. If there are problems you cannot correct, contact Oracle Support Services. The support staff can review the error messages with you to help resolve your problem.

   You can ignore the following error messages when they appear:

   ```
   ORA-00001: Unique constraint violated
   ORA-00942: Table or view does not exist
   ORA-00955: Name is already used by an existing object
   ORA-02289: Sequence does not exist
   ```

7. After making any necessary adjustments rerun migrate_check.sql to verify that all problems have been addressed and corrected.

Note: Be aware that the migrate_check.log file is overwritten every time you run migrate_check.sql. Rename the migrate_check.log file if you want to compare it against files you generate in the future.

Note: Once the Metadata Migration is complete, the FDM database is not accessible from any of the OFS applications until the entire FDM Database Upgrade Process is complete. Any changes to OFSA IDs that you want to perform on the database using the OFS applications must be completed prior to running the Metadata Migration procedure. Once the entire FDM Database Upgrade Process is completed (which includes the generate_upgrade and upgrade steps), the database is accessible by the 4.5 version of the OFS applications.

Run migrate.sql

Follow these steps to run migrate.sql. In this section the OFSA_INSTALL / dbs / <OFSA release> directory is referred to as the database upgrade home directory, or <Upgrade home dir>.

1. In UNIX, change the directory to the <Upgrade home dir> directory.
2. Log into SQL*Plus as SYS using the following command line entry:

   ```
   sqlplus SYS/<password>
   ```

3. Run the cr_owner.sql script, which is in the <Upgrade home dir> / master directory, to grant additional privileges to the database owner. It then prompts you for the name of the OFSA database owner. Prior to running the script, use the SQL*Plus *spool* command to redirect any error messages to a log file.

   ```
   SQL> @master/cr_owner
   ```

   Ignore the following error messages when they appear:

   ```
   ORA-01920: user name <owner_name> conflicts with another user or role
   ```

```
name

ORA-01921: role name <name> conflicts with another user or role name
```

4. Connect as the database owner using the following command line entry:

```
SQL> connect <owner_name>/<password>
```

5. Execute the migrate step of the upgrade procedure. migrate.sql takes the following input parameters:

<Password> This is the password for the database owner, to invoke SQL*Loader.

<Upgrade home dir> This is the full path to the database upgrade home directory as previously described. Do not enter a trailing / character.

An example of a full path follows:

```
/app/ofsa/ofsa4.0-092/dbs/450001010
```

<Sql*Loader executable> This is the command used to execute SQL*Loader. For most Oracle installations, this command is sqlldr.

If you run migrate.sql with no command line parameters, it prompts you for them. It hen prompt you to confirm that the parameters you entered are correct.

```
SQL> @migrate/migrate
```

Or:

```
SQL> @migrate/migrate <Password> <Upgrade home dir> <Sql*Loader
executable>
```

Historical Rates Conversion

If you have not specified the priority for converting Historical Rates IDs in OFSA_TEMP_IRC_45, the migrate procedure displays the following warning:

```
The OFSA_TEMP_IRC_45 table is empty.  If you do not setup the
OFSA_TEMP_IRC_45 table, the DUP will convert your IRCs in the ascending
order of the Historical Rates ID sys_id_num.  It is recommended that
you setup the OFSA_TEMP_IRC_45 table before you run the migration process.

To setup this table exit the migration procedure.

Do you want to exit and setup the OFSA_TEMP_IRC_45 table?(y/n):
```

Refer to Review Migration Requirements for details about the Historical Rates ID conversion.

Select *y* to exit the procedure and setup your Historical Rates ID conversion precedence, or select *n* to continue with the procedure.

Note: If you already populated the OFSA_TEMP_IRC_45 table with your Historical Rates ID conversion precedence, the Migrate procedure continues without prompting for confirmation.

Functional Currency

If you select *n* for the Historical Rates ID conversion prompt, the Migrate then validates your Functional Currency section in OFSA_TEMP_DB_INFO. If you did not correctly specify a Functional Currency in this table, one of the following errors appears:

```
"ERROR! Functional Currency Code not defined in OFSA_TEMP_DB_INFO."
```

```
"ERROR!  Invalid Functional Currency Code in OFSA_TEMP_DB_INFO."
```

Refer to OFSA Database Problem Conditions and Solutions for information on how to resolve this error.

If you have correctly specified a Functional Currency in the OFSA_TEMP_DB_INFO table, the migrate displays the following prompt:

```
You have selected the following Functional Currency Code:
XXX
Do you want to proceed with this installation?(y/n):
```

Select *Y* to continue or *N* to terminate the procedure.

Migrate.sql exits SQL*Plus with an error message under the following conditions:

- You type N at a confirmation prompt
- Any of the input parameters you entered are invalid.
- You have insufficient disk space for the log files and script files that will be produced

Note: Be aware that the migrate.sql automatically re-runs the migrate_check procedure. Any errors that occur are then outputted to the migrate_check.log. The migrate_check.log file is overwritten every time you run migrate.sql. Rename the migrate_check.log file if you want to compare it against files you generate in the future.

6. When **migrate.sql** has finished, exit SQL*Plus and review the migrate log files in the <Upgrade home dir>/log directory for any errors that might have occurred during the migrate.

Caution: The Metadata Migration process alters the password for all users, including the Schema Owner, to OFSA. Individual users cannot log in to any of the OFS applications at this stage of the process. Once the entire upgrade process is complete (which includes running the Upgrade Procedure), all users must alter their individual passwords so that security is not compromised. Oracle recommends that you change the FDM Schema Owner password at this time to ensure a secure environment.

Review Migrate Logs

Migrate.sql creates Procedure logs for messages and errors occurring during the Migrate procedure as well as Conversion logs indicating how your objects and users were migrated into the new FDM 4.5 Metadata.

Note: Review the Procedure logs (migrate1, migrate2, migrate3, and migrate4 log files) first. These log files provide information about the migration process. Any errors that may occur during the process are captured in these files. If you encounter any errors in these log files, contact Oracle Support Services for assistance.

Procedure Logs

Migrate.sql creates the following log files for messages occurring during the procedure. Review these files for any significant errors.

- migrate1.log
- migrate2.log
- migrate3.log
- migrate4.log
- SQL*Loader log files - log files generated by SQL*Loader. There is one log file for each execution of SQL*Loader. In UNIX, you use the **grep** command to display key lines of these files to search for errors.

  An example of a grep command follows:

  ```
  cat 'ls -1t *log'|egrep "(Table|success|errors|Run|null)"
  ```

  The -1t parameter after the ls command is (dash)(number 1)(letter t). Also, the single-quote mark in the statement is the quote mark below the tilda (~) key on the standard keyboard.

Metadata Conversion Logs

The Metadata Conversion logs provide information about how users and objects were migrated into the new 4.5 Metadata structure. Review these log files to verify that your objects and users were migrated correctly.

- rename_objects.log - a list of objects renamed for the 4.5 release. This list includes only internal application tables. The list is provided to inform those users who query against such tables. Refer to Chapter 11, "Upgrading from OFSA 3.5/4.0" for more information on how tables and views are affected by the 4.5 Upgrade Process.
- convert_users.log - a list of Users that exist in the database but are not registered within the FDM metadata. Refer to Chapter 12, "FDM Database Upgrade Process" for more information about how users are affected by the 4.5 Upgrade Process.
- convert_system_info.log - a list of tables that exist in the database but are not registered within the FDM metadata. Refer to Chapter 11, "Upgrading from OFSA 3.5/4.0" for more information about how tables and views are affected by the 4.5 Upgrade Process.
- convert_sys_code_val.log - a list of new tables created to store user-defined Codes and Code descriptions. Before the 4.5 release, this data was stored in SYSTEM_CODE_VALUES. Refer to Chapter 12, "FDM Database Upgrade Process" for more information on how codes and code descriptions are affected by the 4.5 Upgrade Process.

Caution: The Migration procedure registers only FDM 4.5 Metadata for those objects that are identified in the OFSA 3.5/4.0 SYSTEM_INFO table and that exist as valid tables or views in the Oracle RDBMS. The 4.5 Migration procedure does not convert metadata for synonyms or views in the OFSA 3.5/4.0 database that point to objects of a different name.

Running the Upgrade Procedure

When you are ready to run the upgrade procedure be sure to do so from a console at the server's location rather than a remote site.

The Upgrade Procedure includes the following steps:

1. Database Preparation
2. Running the check.sql
3. Executing the Upgrade Procedure
4. Reviewing the Upgrade Logs

Database Preparation

Follow these steps to prepare the database.

1. Unset the ORACLE_PATH and SQL_PATH environment variables so that custom SQL scripts are not executed in place of the upgrade SQL scripts. The command line entries are:

   ```
   unset ORACLE_PATH
   unset SQL_PATH
   ```

   Caution: Unsetting these environment variables is essential. If you do not, you will create problems during the upgrade procedure.

2. Verify that your $ORACLE_SID points to the correct ORACLE instance on the server.

3. Remove any tables in the current schema that have the prefix O_ (the letter *O* followed by an underscore).

   The O_ prefix is reserved for the OFSA database upgrade procedure. If you do not remove these tables from your system, you receive a problem warning message in the check.log file. Make sure that the following minimum space is available before you begin:

SYSTEM tablespace	10 MB (minimum) assuming initjvm.sql already run
DATA_TS tablespace	100 MB (minimum)
INDEX_TS tablespace	50 MB (minimum)
UNIX file system	10 MB for log files and temporary files

> **Note:** These freeespace requirements pertain to completing the generate_upgrade step only. The FDM Database Upgrade Process then reports the freespace requirements to complete the procedure in the header of the update.sql script file.

4. The upgrade process requires a sort_area_size of at least 20 MB.

   If your sort_area_size is more than 20 MB, do not decrease it. If you need to increase your sort_area_size for the upgrade process, record the current value of the sort_area_size parameter (found in the init<dbname>.ora file) and then increase it to at least 20 MB.

   At the end of the procedure this parameter can be restored to its original size.

5. Shut down the Oracle listener.

6. Shut down and restart the database.

   This guarantees that no one but the System Administrator is accessing the database during the upgrade procedure and ensures that any new initialization parameters take effect.

Running check.sql

The upgrade check is a process that enables you to identify and correct errors in your database before you execute the upgrade procedure. The pre-upgrade check

produces a text file listing all database problems that can cause the upgrade procedure to fail. Running this process does not alter the OFSA database.

Follow these steps to run check.sql. In this section the OFSA_INSTALL/dbs/<OFSA release> directory is referred to as the database upgrade home directory, or <Upgrade home dir>.

1. In UNIX, change the directory to the <Upgrade home dir> directory.
2. Log into SQL*Plus as SYS using the following command line entry:

   ```
   sqlplus SYS/<password>
   ```

3. Connect as the database owner using the following command line entry:

   ```
   SQL> connect <owner_name>/<password>
   ```

   Execute the upgrade check process using the following command line entry:

   ```
   SQL> @check <Upgrade home dir>
   ```

   <Upgrade home dir> is the full path to the database upgrade home directory. Do not enter a trailing / character.

   An example of a full path follows:

   ```
   /app/ofsa/ofsa4.0-092/dbs/450001010
   ```

4. Review the check.log script file, created in the <Upgrade home dir>/log directory, for any problem conditions.

   If error conditions appear in the check.log script file, refer to "OFSA Database Problem Conditions and Solutions" to resolve them. If there are problems you cannot correct, contact Oracle Support Services. The support staff can review the error messages with you to help resolve your problem.

   You can ignore the following error messages when they appear:

   ```
   ORA-00001: Unique constraint violated
   ORA-00942: Table or view does not exist
   ORA-00955: Name is already used by an existing object
   ORA-02289: Sequence does not exist
   Error during - ALTER TABLE OFSA_INSTALL_GROUPS ADD CONSTRAINT
   Error during - DROP TABLE or DROP SEQUENCE
   ```

5. After making any necessary adjustments rerun check.sql to verify that all problems have been addressed and corrected.

Note: Be aware that the check.log file is overwritten every time you run check.sql. Rename the check.log file if you want to compare it against files you generate in the future.

You can begin the upgrade procedure when no problems appear in the check.log file.

Executing the Upgrade Procedure

Follow these steps to execute the upgrade procedure. In this section the OFSA_INSTALL / dbs / <OFSA release> directory is referred to as the database upgrade home directory, or <Upgrade home dir>.

1. In UNIX, change directory to the <Upgrade home dir> directory.
2. Log into SQL*Plus as SYS using the following command line entry:

   ```
   sqlplus SYS/<password>
   ```

3. Log into SQL*Plus as the database owner using the following command line entry:

   ```
   SQL> connect <owner_name>/<password>
   ```

4. Execute the generate_upgrade step of the upgrade procedure. generate_upgrade.sql takes the following input parameters:

   <Password> This is the password for the database owner, to invoke SQL*Loader.

   <Upgrade home dir> This is the full path to the database upgrade home directory as previously described. Do not enter a trailing / character.

   An example of a full path follows:

   ```
   /app/ofsa/ofsa4.0-092/dbs/400001010
   ```

   <Sql*Loader executable> This is the command used to execute SQL*Loader. For most Oracle installations, this command is sqlldr.

   If you run generate_upgrade.sql with no command line parameters, it prompts you for them. It then prompts you to confirm that the parameters you entered are correct.

```
SQL> @generate_upgrade
```

Or:

```
SQL> @generate_upgrade <Password> <Upgrade home dir> <Sql*Loader
executable>
```

Generate_upgrade.sql exits SQL*Plus with an error message under the following conditions:

- You type N at the confirmation prompt
- Any of the input parameters you entered are invalid
- You have insufficient disk space for the log files and script files that will be produced

---

Note: The generate_upgrade step of the FDM Database Upgrade Process may require significant time (several hours or more) in order to complete depending upon your database environment and the number of objects in your database instance.

---

5. After **generate_upgrade.sql** finishes, exit SQL*Plus and review the log files and the **update.sql** script file. You can use the sample grep command provided in Reviewing the Upgrade Logs to complete the review of the log files.

   The generate_upgrade process produces the following log files located in the <upgrade home dir>/log directory.

   - generate_upgrade.log

   Review the generate_upgrade.log files in detail. Ignore error messages resulting from the following processes:

   - Attempting to drop or alter an object that does not exist
   - Attempting to create an object that exists already
   - Attempting to insert data that exists already (unique constraint violations)

   In **generate_upgrade.log** check to make sure that none of the package creations or procedure executions generated any error messages. If you find significant errors, contact Oracle Support Services before continuing with the upgrade procedure.

   update.sql is created by the **generate_upgrade** script and is used by the **upgrade** script. This script performs the structural changes required by this upgrade. If you ran the upgrade check process and corrected all problems

reported, an upgrade script is created. The generate_upgrade script creates the **update.sql** script in the <upgrade home dir> directory.

> **Note:** It is important that you review the database space report in the update.sql file to verify that your tablespaces have the required free space before you proceed. The amount of free space reported is an estimate, however, and in some cases may vary significantly from the amount required.

It is important that you review the database space report in the update.sql file to verify that your tablespaces have the required free space before you proceed. The amount of free space reported is an estimate, however, and in some cases may vary significantly from the amount required.

This report is intended to alert you to potential space problems before you proceed with the upgrade.

> **Caution:** The update.sql adds a new column, ISO_CURRENCY_CD, to Instrument tables in order to enable multi-currency functionality. This column is required by the OFS applications and is defined as NOT NULL. Because it is defined as NOT NULL, the upgrade process updates this column on all instrument tables with a default currency as defined in `OFSA_TEMP_DB_INFO`. Depending upon the number of records present in each instrument table, this update may require a significant amount of time for completion.

By default, update.sql assumes that you have a data tablespace named DATA_TS and an index tablespace named INDEX_TS. If you do not want new tables and indexes to be created in these tablespaces, you need to edit the upgrade script after it has been created.

Review the storage clauses for each of the tables and indexes created in this script and modify them as appropriate for your implementation.

Note: Because the upgrade log files generated in the next step are spooled by SQL*Plus, the full text of the SQL*Loader executions called from upgrade.sql do not appear in the log file. If a SQL*Loader step fails, you cannot detect it in the upgrade log files. Use the following UNIX script command to capture all output from the run of upgrade.sql, including the SQL*Loader executions:

```
script -a my_upgrade.log
sqlplus <owner_name>/<password>
SQL>@upgrade <Password> <Upgrade home dir> <Sql*Loader
executable>
SQL> exit
Exit
```

---

6. Log into SQL*Plus as the database owner.

   ```
   sqlplus <owner_name>/<password>
   ```

7. Execute the upgrade step of the upgrade procedure.

   Upgrade.sql takes the same parameters as generate_upgrade.sql, and you can either pass them in using the command line, or allow the script to prompt for them. Upgrade.sql calls update.sql and then rebuilds FDM metadata and reloads FDM seeded data.

   ```
   SQL> @upgrade <Password> <Upgrade home dir> <Sql*Loader executable>
   ```

   Caution: Do NOT execute the update.sql script by itself. Instead, run the upgrade.sql script to complete the procedure.

   ---

8. When upgrade.sql has finished, exit SQL*Plus and review the upgrade log files in the <Upgrade home dir>/log directory for any errors that might have occurred during the upgrade. Refer to Reviewing the Upgrade Logs for detailed instructions on how to review and resolve errors generated by the upgrade process.

9. Log on to SQL*Plus as the database owner.

   ```
   sqlplus <owner_name>/<password>
   ```

10. Execute grant_all:

```
SQL> set serveroutput on
SQL> execute ofsa_dba.grant_all('OFSA','COMMAND_LINE');
```

If you encounter any errors in running this procedure, refer to Chapter 16, "FDM Object Management".

11. Restore the sort_area_size parameter (in **init<dbname>.ora**) to its original value if you have changed it.

12. Shut down and restart the database.

13. Restart the Oracle listener.

Note: Once the Upgrade Procedure is completed, you still need to install and configure the Oracle Discoverer Business Areas and Workbooks. Refer to Chapter 13, "Installing and Configuring Discoverer" for information on this procedure.

Caution: All passwords for FDM-seeded Internal roles are set to XXADCFGZ. Oracle recommends that you change these role passwords immediately after completing the FDM upgrade procedure. To do so, use the Change Password functionality within the FDM Administration application.

Caution: OFS application multiprocessing settings in FDM 4.5 are no longer specified in the server ini files. Instead, they are designated in the database. Because of this, all OFS application multiprocessing settings revert to the default after the FDM upgrade process is complete. Refer to the Chapter 19, "OFSA Multiprocessing" for more information.

Reviewing the Upgrade Logs

The final step of the FDM database upgrade process creates several log files requiring review. These files are categorized as follows:

- Primary Log Files
- Row Count Log File
- ID Conversion Log Files
- SQL Loader Log Files
- Internal Log Files (safe to ignore)

Primary Log Files

The Primary Log Files record all of the DDL statements executed by the upgrade, as well as migration logic not related to ID Conversions. The 2 log files recording this information are:

- **<xxxxxxxxx>.log**
- **<xxxxxxxxx>_part_two.log**

where xxxxxxxxx is the FDM database release. For example: 450000037.log and 450000037_part_two.log.

It is important to review all of these log files when the upgrade step is complete. Errors and messages in these log files are categorized as follows:

- Acceptable Errors - these errors can be ignored with no consequences.
- Constraint Errors - these errors can, in most cases, be resolved after the upgrade process is complete.
- Fatal Errors - these errors require running the upgrade process again from the beginning.
- Table Classification Validation Messages - these messages indicate objects that failed Table Classification requirements.

Acceptable Errors

In reviewing these files, you can ignore the following types of errors:

- Errors resulting from dropping an object that does not exist (including ALTER statements for dropping constraints)
- Errors resulting from attempting to create an object that already exists
- Errors resulting from attempting to add a constraint that already exists on a table

- Compilation errors for the ITG_RESOLVE_RANGE function. This function is used for integration with the Oracle General Ledger product and is therefore only operational when linked to an Oracle General Ledger database.
- Oracle error 4054 - database link OTBFS does not exist during the adding of the package body for the OFSA_OMM_TO_VFS package. This package is used for the integration with the Oracle Telebusiness product, and is therefore only operational when linked to an Oracle Telebusiness database.

Constraint Errors

FDM 4.5 implements both primary key and foreign key constraints. Because the OFSA 3.5/4.0 database did not implement such constraints, it is possible that bad data from these previous versions may cause errors in enabling constraints in FDM 4.5.

The FDM upgrade process creates a table named OFSA_MSI_EXCEPTIONS and records any rows violating database constraints. This table identifies the table and constraint that is violated, as well as the row_id of the record causing the violation. After the upgrade process is complete, review this table for any possible violations. If any violations do exist, follow these guidelines to resolve them:

1. Identify constraint violations in OFSA_MSI_EXCEPTIONS
2. Update the records in the designated table name appropriately using the specified row_id values. Refer to the Oracle catalogs (such as USER_CONS_ COLUMNS and USER_CONSTRAINTS) to identify the constraint definition.
3. Use the Alter-Table-enable-constraint syntax to activate the constraints.

Refer to the appropriate Oracle RDBMS documentation regarding the appropriate SQL syntax for enabling constraints as well as general information about Oracle constraints

If you encounter constraint errors that you are unable to resolve, contact Oracle Support Services for assistance.

Fatal Errors

If you encounter any significant errors in any of the log files (or if you are not sure if a particular error is fatal or not), contact Oracle Support Services. The support staff can review the messages with you and assist in making the necessary corrections.

Table Classification Validation Messages

FDM 4.5 classifies objects (tables and views) for use with OFS applications. Each Table Classification identifies a specific purpose for which an assigned table or view is allowed to be used. Each of these Table Classifications has requirements that must be met in order for a table or view to receive that classification.

Because the Table Classification concept is a new one for 4.5, the FDM database upgrade procedure maps tables and views from OFSA 3.5/4.0 to the appropriate Table Classifications in 4.5. However, some objects mapped by this procedure may fail the validation requirements for one or more Table Classifications. The classification failure messages are logged in the **<xxxxxxxxx>_part_two.log** file.

Review the Table Classifications validation messages in this file. For any Table Classification failures, if you are not using the specified object for that purpose, you do not need to resolve the indicated problem. Only when you intend to use the object for a particular Table Classification for which it failed do you need to do anything.

If you do intend on using an object for a Table Classification for which it failed, you need to modify the object so that it meets the Table Classification requirements. Once the object is modified, use the FDM Administration application to re-register the object and then manually assign the Table Classification to the object.

Note: All of the Table Classification validations for Instrument tables fail if you have altered column definitions or attributes for columns reserved for FDM. FDM allows you to change only the data length, precision, and scale. If you have changed one of these attributes for a column, you need to update the OFSA_COLUMN_REQUIREMENTS table to match your altered definition. If you have altered any other attributes of a column, you must modify the column to match the requirements specified in OFSA_COLUMN_REQUIREMENTS. Refer to Chapter 16, "FDM Object Management" for details on how to resolve such occurrences.

Refer to Chapter 16, "FDM Object Management" details on the specific Table Classification requirements.

Row Count Log File

The **upgrade1.log** file contains a report comparing record counts for FDM 4.5 tables to record counts in the previous version. With some exceptions, when a table is recreated to alter its definition to the new FDM 4.5 structure, the upgrade process

records the record count for both the original table and the new table. Review this log to verify that the 4.5 upgrade process preserves data from your OFSA 3.5/4.0 database.

The exceptions for which the upgrade process does not preserve data are as follows:

- All _DAT tables
- OFSA_PROCESS_CASH_FLOWS
- OFSA_STP

The data in the _DAT tables is always reloaded during processing and is not accessed directly by any OFS operations. The data in the OFSA_PROCESS_CASH_FLOWS and OFSA_STP tables is for audit purposes only. The OFSA 3.5/4.0 data for these tables is not valid for FDM 4.5.

Disparity in row counts in OFSA ID tables is caused by the ID Conversion routines. Refer to the log files for these conversions to verify that all of your OFSA IDs were migrated into the FDM 4.5 database structure as expected. Disparity in rows counts is also caused by changes in the seeded data. The seeded data for the following tables is comprised of fewer rows than in previous releases:

- OFSA_INDEX_STORAGE_DEFAULTS
- OFSA_TABLE_STORAGE_DEFAULTS

SQL Loader Logs

Review the log files generated by SQL*Loader. There should be one log file for each execution of SQL*Loader. In UNIX, you can use the **grep** command to display key lines of these files to search for errors.

An example of a grep command follows:

```
cat ‘ls -1t *log‘|egrep "(Table|success|errors|Run|null)"
```

If you followed the procedures in the previous step, then review the script log file thoroughly to verify that every step in the upgrade was successful. The -1t parameter after the ls command is (dash)(number 1)(letter t). The single-quote mark in the statement is the quote mark below the tilda (~) key on the standard keyboard.

ID Conversion Logs

The ID Conversion logs provide information regarding the conversion of OFSA IDs to the new 4.5 FDM database structure. Review the following ID Conversion logs to identify any IDs that encountered problems:

- alloc_id_conv.log
- fcast_rate_id_conv.log
- leaf_charc_conv.log
- rate_manager_conv.log
- trans_strategy_conv.log

Note: The Forecast Rate ID conversion log file (fcast_rate_id_conv.log) may indicate that rates exist in RATES_FORECAST for scenario_nums that do not exist in RATES_SCENARIO. This situation occurs because OFSA 3.5/4.0 did not properly delete data from RATES_FORECAST when a user deleted a Forecast Rates ID. Messages relating to this situation in the fcast_rate_id.log file can be ignored.

Internal Log Files (safe to ignore)

Ignore the contents of the msi_time_xxxxxx.log file as it is an internal log file. You do not need to review this file.

Password Encryption Changes

Security and password encryption is significantly different in version 4.5 than it was in previous OFSA versions. Previously, all user passwords were encrypted so that the user was not aware of the true Oracle RDBMS password. Users were allowed to access the database only through one of the OFS applications (such as Risk Manager or Performance Analyzer). A second account with separate, more restrictive privileges would then be created for the user to allow for reporting and other query operations using other Oracle compatible tools (such as Oracle Discoverer or SQL*Plus). This mechanism ensured that privileges that users required during application operations were not available to them in other less controlled database sessions.

The security implementation for FDM version 4.5 is significantly different from this. Because FDM supports the concept of password protected *Internal* roles (roles enabled only within one of the OFS applications) and *External* roles (roles available for any database session), it is no longer necessary to encrypt user passwords. The Metadata Migration of the 4.5 Database Upgrade Process resets all user passwords, including that of the OFSA Schema Owner, to OFSA. Once the upgrade process is

complete, it is imperative for all users to alter their individual passwords so that security is not compromised.

Passwords for Internal roles seeded in the FDM database are set to XXADCFGZ by the 4.5 database upgrade process. The passwords for these roles should also be altered after the upgrade is complete so that database security is not compromised.

OFSA Database Problem Conditions and Solutions

While upgrading your database, you may encounter problems that halt the procedure. Such problems are identified during the process by both the Metadata Migration check and the Database Upgrade Check. The information provided in this section describes how to resolve each of these problems so that you may proceed with the upgrade.

The problem conditions listed here are described in detail in the following sections:

ID Errors

- Client ids in seeded ID range
- Leaf Characteristics ID or Transaction Strategy ID has incorrect number of rows
- TP Process <sys_id_num> is Transfer Pricing ID that has been deleted.

General Errors

- Client data in the ofsa_correction_proc_msg_cd data range
- Existing Role conflicts with a seeded Role
- Functional Currency not defined or invalid in OFSA_TEMP_DB_INFO
- INIT.ora parameters not correct
- Invalid data in OFSA_TEMP_IRC_45
- o_ tables have been found

Leaf Errors

- Client data in the detail_elem (or ofsa_detail_elem) seeded data range
- Client data in the leaf_desc (or ofsa_leaf_desc) seeded data range
- Column_name is null in OFSA_DETAIL_ELEM
- Duplicate column_name values in ofsa_detail_elem

User and User Group Errors

- Identical User or User Group names
- User running the upgrade must be the FDM Schema Owner
- User conflicts with seeded Recipient Name or ID Folder
- User conflicts with User Group to be created
- User conflicts with Security Profile to be created
- User or Group in CATALOG_OF_USERS not uppercase
- User <username> in HARV_USER not uppercase
- <group_name> not a valid User Group

SYSTEM_CODE_VALUES Errors

- Alpha values found in numeric columns
- Column_name in SYSTEM_CODE_VALUES not uppercase
- Instrument values in SYSTEM_CODE_VALUES not uppercase
- Duplicate values in SYSTEM_CODE_VALUES
- NULL values in SYSTEM_CODE_VALUES

SYSTEM_INFO Errors

- Duplicate DISPLAY_NAME values in SYSTEM_INFO
- Null values found in SYSTEM_INFO columns
- Tables in SYSTEM_INFO have the same display_name
- Table or Column Name in SYSTEM_INFO not uppercase

ID Errors

Client IDs in seeded ID range

The following error message appears:

"Update cannot proceed, client id's have been found in seeded ID range"

Problem:

The database upgrade procedure has found rows in the CATALOG_OF_IDS table where SYS_ID_NUM is between 80,000 and 100,000 and GROUP_NAME is not <TSER> or <OFSA>. This range of SYS_ID_NUM values is reserved for data seeded by the database upgrade process.

```
SELECT sys_id_num, id_desc_short, id_type, group_name
FROM catalog_of_ids
WHERE sys_id_num >= 80000
AND sys_id_num <= 100000
AND group_name <> 'TSER'
AND group_name <> '<OFSA>';
```

Solution:

Log in to the appropriate OFSA product and delete the IDs identified by the SYS_ID_NUM values returned by the SELECT statement. To save an ID with appropriate SYS_ID_NUM values, open the ID and perform SAVE AS prior to deleting it. Rerun the SELECT statement. If any offending rows still remain, delete them from CATALOG_OF_IDS so that the database upgrade can continue.

Leaf Characteristics ID or Transaction Strategy ID has incorrect number of rows

One or more of the following error messages appears:

```
"ERROR!  Transaction Strategy ID <sys_id_num> has incorrect number of rows
for one or more leaves and transactions."
```

```
ERROR!  Leaf Characteristics ID <sys_id_num> has incorrect number of rows
for one or more leaves and transactions.
```

Problem

A Transaction Strategy ID or Leaf Characteristics ID is invalid.

Use the following SQL to identify invalid Leaf Characteristics IDs:

```
SELECT DISTINCT a.sys_id_num,
b.id_desc_short||'.'||b.group_name id_name
FROM idt_tm_details a, catalog_of_ids b
WHERE b.id_type  = 309
AND b.sys_id_num = a.sys_id_num
GROUP BY a.sys_id_num,a.leaf_node,b.id_desc_short,b.group_name
HAVING COUNT(*) <> 42;
```

Use the following SQL to identify invalid Transaction Strategy IDs:

```
SELECT DISTINCT a.sys_id_num,
b.id_desc_short||'.'||b.group_name id_name
FROM idt_tm_details a, catalog_of_ids b
WHERE a.sys_id_num = b.sys_id_num
AND b.id_type=306
GROUP BY a.sys_id_num,a.leaf_node,a.leaf_num_id,b.id_desc_short,
b.group_name
HAVING COUNT(*) <> 50;
```

Solution

Remove the specific IDs using the 4.0 Risk Manager application.

TP Process <sys_id_num> is using invalid Transfer Pricing ID

One of the following error messages appears:

```
"ERROR!  TP Process ID <sys_id_num> is using a Transfer Pricing ID that has
been deleted."
```

Problem

A TP Process ID exists that is referencing an invalid or non-existent Transfer Pricing ID.

Use the following SQL to identify invalid TP Process IDs:

```
SELECT sys_id_num FROM idt_process
WHERE id_type=200 AND process_sys_id NOT IN
(SELECT sys_id_num FROM idt_transfer_price
WHERE sys_id_num IN
(SELECT sys_id_num FROM catalog_of_ids WHERE id_type=200));
```

Solution

If the sys_id_num identified by the SQL does not exist in CATALOG_OF_IDS, then you must manually delete the records from IDT_PROCESS for that identified sys_id_num.

If the identified sys_id_num does exist in CATALOG_OF_IDS, then it can be removed from the database using the 4.0 Transfer Pricing application. You can identify the appropriate ID to delete by running the following SQL:

```
SELECT id_desc_short, group_name
FROM CATALOG_OF_IDS
WHERE sys_id_num = <sys_id_num>;
```

General Errors

Client data in the ofsa_correction_proc_msg_cd data range

The following error message appears:

"Update cannot proceed, there may be client data in the seeded data range of system_error_code."

Problem:

The database upgrade process has found the incorrect number of rows in OFSA_CORRECTION_PROC_MSG_CD where the ERROR_CODE>= 9,000 or ERROR_CODE=0.

This range of system_error_code is reserved for data seeded by the database upgrade process.

```
IF current_db_version >= 300.000002 AND
    current_db_version <350.000000 THEN
    SELECT (count(*)-84)
    INTO system_count
    FROM system_error_code
    WHERE error_code >= 9000;

ELSIF current_db_version >= 350.000000 THEN
    SELECT (count(*)-85)
    INTO system_count
    FROM system_error_code
    WHERE error_code >= 9000;
        OR error_code = 0
    END IF;

    IF (system_count <=0) THEN
        There is no problem

    ELSE
        There is client data in the seeded range of the system_error_code
        table.  The offending rows must be deleted before proceeding with
        the upgrade.

    END IF;
```

Solution:

Archive the existing table, perform the upgrade and then compare the new table to the old table to identify the offending rows. Delete or move all offending data in this range. Contact Oracle Support Services before attempting to solve this problem.

Existing Role conflicts with a seeded Role

The following message appears:

```
Existing Role conflicts with a seeded Role.
```

Problem

The designated Role is a reserved Role for the FDM database. FDM reserves specific Role names for application use.

Solution:

Drop the identified Role and re-create it with a different name.

Functional Currency not defined or invalid in OFSA_TEMP_DB_INFO

One of the following error messages appears:

```
"ERROR! Functional Currency Code not defined in OFSA_TEMP_DB_INFO."

"ERROR!  Invalid Functional Currency Code in OFSA_TEMP_DB_INFO."
```

Problem

The upgrade process references the currency_cd value from the OFSA_TEMP_DB_INFO table to populate the ISO_CURRENCY_CD field in the database with an appropriate currency code. The OFSA_TEMP_DB_INFO table is created by the Metadata Migration Check to allow the user to specify this default value for the upgrade. Because the OFSA_TEMP_DB_INFO table is created by the Metadata Migration Check, you always need to populate it with a valid currency code.

Solution

Insert a single record into the OFSA_TEMP_DB_INFO table. For example:

```
INSERT INTO ofsa_temp_db_info
values ('USD');
```

In the case where there already is a record but you have received the Invalid Functional Currency Code error, make sure that the value is a valid Currency Code. A list of valid Currency Codes is available in Appendix A, "Functional Currencies".

Note: The FDM Database Upgrade Process requires that the OFSA_TEMP_DB_INFO table have only one row.

INIT.ora parameters not correct

One of the following error messages appears:

```
"ERROR!  INIT.ora compatible parameter must be 8.1.5."

"ERROR!  INIT.ora max_enabled_roles parameter must be greater than or equal
to 60."

"ERROR!  INIT.ora dml_locks parameter must be greater than or equal to 200."
```

Problem

The following parameters are required for the INIT.ora file for the instance:

compatible = 8.1.6

max_enabled_roles>=60

dml_locks >= 200

Solution

Correct the INIT.ora file for the instance. Then shutdown the instance and restart it.

Invalid data in OFSA_TEMP_IRC_45

One of the following error / warning messages appears:

"ERROR! Invalid Base Currency CD <currency_cd> in OFSA_TEMP_IRC_45"

"ERROR! Invalid rates_sys_id <sys_id_num> in OFSA_TEMP_IRC_45"

"OFSA_TEMP_IRC_45 table is empty. If you do not setup OFSA_TEMP_IRC_45 the upgrade will convert IRC in the ascending order of the History Rate ID sys_id_num."

Problem

The upgrade process uses the currency_cd value from the OFSA_TEMP_IRC_45 table to perform data Historical Rates data conversions. The OFSA_TEMP_IRC_45 table is created by the Metadata Migration Check to allow the user to specify the values used for this conversion. In this case the data inserted into this table for the data conversions is invalid.

Solution

The currency_cd column in OFSA_TEMP_IRC_45 is used to store the Base Currency Code. A list of valid Currency Codes is available in Appendix A, "Functional Currencies". For example, the ISO_CURRENCY_CD USD is for American dollars.

The rates_sys_id column in OFSA_TEMP_IRC_45 is the Historical Rates ID that is converted during the upgrade to the new 4.5 structure. If you do not specify one or more valid Historical Rates IDs in the rates_sys_id column of this table prior to running the upgrade (that is, there are no rows in this table), the upgrade converts all of the Historical Rates IDs in the database. For more information, see Historical Rates Conversion.

o_ tables have been found

The following error message appears:

```
"Update cannot proceed, o_ tables have been found."
"Please remove all o_ tables and re-run this step."
```

Problem:

The database upgrade process has found tables owned by the database owner that start with o_. This prefix is reserved for the FDM database upgrade procedure.

```
SELECT table_name
FROM user_tables
WHERE table_name like 'o\_%' ESCAPE '\'
OR table_name like 'O\_%' ESCAPE '\';
```

Solution:

All tables with this prefix condition need to either be renamed or dropped from the OFSA database owner schema before the database upgrade can continue.

Leaf Errors

Client data in the detail_elem (or ofsa_detail_elem) seeded data range

One of the following error messages appears:

```
"Update cannot proceed, there may be client data in the seeded data range of detail_elem."

Update cannot proceed, there may be client data in the seeded data range of ofsa_detail_elem."
```

Problem:

The database upgrade procedure has found the incorrect number of rows in DETAIL_ELEM or OFSA_DETAIL_ELEM table where the LEAF_NODE is less than 10,000. This range of LEAF_NODE values is reserved for data seeded by the database upgrade process.

If you are running the Metadata Migration Check procedure, the problem is for the DETAIL_ELEM table. If you are running the Database Upgrade Check, the problem is for the OFSA_DETAIL_ELEM table.

The acceptable count of values for DETAIL_ELEM and OFSA_DETAIL_ELEM is based upon the VERSION value in the OFSA_VERSION table where APP_NAME='Database'. For all SQL statements, substitute OFSA_DETAIL_ELEM for DETAIL_ELEM if you are validating these counts after the Metadata Migration is complete:

```
For VERSION <= 350.000124
    SELECT (COUNT(*)-157) FROM detail_elem
    WHERE leaf_node < 10000;
For VERSION <= 400.000020
    SELECT (COUNT(*)-163) FROM detail_elem
    WHERE leaf_node < 10000;
For VERSION < 400.101067
    SELECT (COUNT(*)-145) FROM detail_elem
    WHERE leaf_node < 10000;
For VERSION < 450.000000
    SELECT (COUNT(*)-165) FROM detail_elem
    WHERE leaf_node < 10000;
Otherwise
    SELECT (COUNT(*)-181) FROM detail_elem
    WHERE leaf_node < 10000;
```

Solution:

The offending rows in DETAIL_ELEM or OFSA_DETAIL_ELEM are user-defined, financial element values incorrectly defined within the reserved range. Recreate these financial element values in Leaf Setup with new leaf values outside of the seeded data range (>10000) and update all data tables with the new values. Then delete the inappropriate data in the reserved range.

Missing financial elements in the seeded data range are automatically re-populated by the database upgrade process.

Client data in the leaf_desc (or ofsa_leaf_desc) seeded data range

The following error message appears:

```
Update cannot proceed,  there may be client data in the seeded data range of leaf_desc.
```

```
Update cannot proceed,  there may be client data in the seeded data range of ofsa_leaf_desc.
```

Problem:

The database upgrade process has found the incorrect number of rows in LEAF_DESC or OFSA_LEAF_DESC where the LEAF_NUM_ID = 0 and the LEAF_NODE is less than 10,000. This range of LEAF_NODE values is reserved for data seeded by the database upgrade procedure

If you are running the Metadata Migration Check procedure, the problem is for the LEAF_DESC table. If you are running the Database Upgrade Check, the problem is for the OFSA_LEAF_DESC table.

The acceptable count of values for LEAF_DESC and OFSA_LEAF_DESC is based upon the VERSION value in the OFSA_VERSION table where APP_NAME='Database'. For all SQL statements, substitute OFSA_LEAF_DESC for LEAF_DESC if you are validating these counts after the Metadata Migration is complete:

```
For VERSION <= 350.000124
    SELECT (COUNT(*)-157) FROM leaf_desc
    WHERE leaf_num_id = 0 AND leaf_node < 10000;
For VERSION <= 400.000020
    SELECT (COUNT(*)-163) FROM leaf_desc
    WHERE leaf_num_id = 0 AND leaf_node < 10000;
For VERSION < 400.101067
    SELECT (COUNT(*)-145) FROM leaf_desc
    WHERE leaf_num_id = 0 AND leaf_node < 10000;
For VERSION < 450.000000
    SELECT (COUNT(*)-165) FROM leaf_desc
    WHERE leaf_num_id = 0 AND leaf_node < 10000;
Otherwise
    SELECT (COUNT(*)-181) FROM leaf_desc
    WHERE leaf_num_id = 0 AND leaf_node < 10000;
```

Solution:

The rows in LEAF_DESC causing the problem are user-defined, financial element values incorrectly defined within the reserved range. Recreate these financial

element values in Leaf Setup with new leaf values outside of the seeded data range (>10000) and update all client data tables with the new values. Then delete the data from the reserved range.

Column_name is null in ofsa_detail_elem

The following error message appears:

```
ERROR: Update cannot proceed,column_name is null in OFSA_DETAIL_ELEM.
```

Problem:

The column_name in OFSA_DETAIL_ELEM is the name of the output column for any Transformation processing for the designated Financial Element. Each Financial Element in OFSA_DETAIL_ELEM must have an output column_name value assigned to it.

To identify the null column_name values in OFSA_DETAIL_ELEM, run the following query in SQL*Plus:

```
SELECT leaf_node, column_name
FROM ofsa_detail_elem
WHERE column_name IS NULL;
```

Solution:

Update the column_name values in OFSA_DETAIL_ELEM so that each Financial Element ID (leaf_node) has a unique column_name.

Duplicate column_name values in ofsa_detail_elem

The following error message appears:

```
ERROR: Update cannot proceed, duplicate column_name values in OFSA_DETAIL_
ELEM.
```

Problem:

The column_name in OFSA_DETAIL_ELEM is the name of the output column for any Transformation processing for the designated Financial Element. Each Financial Element in OFSA_DETAIL_ELEM must have a unique output column_name value assigned to it.

To identify the duplicate column_name values in OFSA_DETAIL_ELEM, run the following query in SQL*Plus:

```
SELECT column_name, count(*)
FROM ofsa_detail_elem
GROUP BY column_name HAVING COUNT(*) > 1;
```

Solution:

Update the column_name values in OFSA_DETAIL_ELEM so that each Financial Element ID (leaf_node) has a unique column_name.

User Errors

Identical User or User Group names

One of the following error messages appears:

```
"ERROR!  Identical User Group Names due to trailing spaces."

"ERROR!  Identical User or User Group name <name>"
```

Problem

OFSA 4.0 allowed a User Name and a Group Name to be identical. The FDM 4.5 database does not allow this. In 4.5, both Users and User Groups are categorized as *Recipients*, and must have distinct names.

Use the following SQL to identify 4.0 Users or Groups with duplicate names:

```
SELECT rtrim(login_name), count(*)
FROM catalog_of_users
GROUP BY rtrim(login_name)
HAVING count(*)>1
```

Solution

Rename or remove the duplicate Users and User Groups using the 4.0 System Administration application.

User running the upgrade must be the FDM schema owner

The following error message appears:

```
"Update cannot proceed, The username used to login to SQL*Plus for running
the upgrade must be the same as the database owner recorded in the
database."
```

Problem

The username used to run the database upgrade procedure must be identified in the OFSA database as the database owner. To identify the username designated as the OFSA database owner, execute the following SQL statement:

```
SELECT login_name
FROM catalog_of_users
WHERE user_owner=1;
```

Solution

Only one record should exist in the CATALOG_OF_USERS table where the USER_OWNER=1. In addition, the LOGIN_NAME designated in CATALOG_OF_USERS where USER_OWNER=1 must be the schema name that owns all of the OFSA tables and objects.

If there is no record in CATALOG_OF_USERS where USER_OWNER=1, contact Oracle Support Services for information on how to create a correct entry. If there is a record in CATALOG_OF_USERS where USER_OWNER=1 but the LOGIN_NAME is different than the schema name that owns all of the OFSA tables and objects, then you must update the LOGIN_NAME with the correct username prior to continuing with the upgrade. However, if such a situation exists, it is possible that other dependent OFSA data for the username must also be corrected. Before continuing with the upgrade process, contact Oracle Support Services for assistance.

User conflicts with seeded Recipient Name or ID Folder

One of the following error messages appears:

"ERROR! Existing User <username> conflicts with a seeded Recipient Name."

"ERROR! User in ofsa_results_table_types conflicts with a seeded Recipient Name"
"ERROR! User in ofsa_users_table_tracking conflicts with a seeded Recipient Name"
ERROR! Existing User Group or User named ALL, OFSA or DELETED_USERS'.

Problem

The username is the same name as a Recipient Name reserved by FDM. All reserved FDM Recipient Names are identified by an FDM_ prefix. ID Folders reserved by FDM include ALL, OFSA, and DELETED_USERS.

Solution

FDM reserves certain names for the OFS applications. The specific 4.0 User Name or Group must be deleted from the database using the OFSA 4.0 System Administration application.

See Chapter 14, "FDM Security" for information about recipient names reserved by FDM.

User conflicts with User Group to be created

The following error message appears:

```
"ERROR!  Existing User <username> conflicts with User Group to be created
during migration."
```

Problem

The username is the same name as a User Group Name that is created by the Metadata Migration process. The Metadata Migration process migrates a 4.0 Group into the following entities:

Example: 4.0 Group name is RM_USERS

Entity Type	Entity Name
User Group	G_RM_USERS
Security Profile	S_RM_USERS
ID Folder	RM_USERS

Because the Group concept in OFSA 4.0 embodied the functionality for User Groups, Security Profiles and ID Folders, the Metadata Migration process creates separate entities from a single Group. The problem identified by the Migrate check is that there is a User or Group Name that is the same name as a User Group to be created by the Migrate. In the example, there is a 4.0 Group named RM_USERS and G_RM_USERS.

The following SQL identifies any conflicts that might exist for User Groups to be created by the migrate:

```
SELECT username FROM all_users
      WHERE username IN
        (SELECT 'G_' || DECODE(login_name,'<ALL>','ALL',
                                          '<OFSA>','OFSA',
                                          login_name)
         FROM catalog_of_users
         WHERE type_cd = 2);
```

Solution

Rename or remove the Groups in 4.0 which cause this problem.

User conflicts with Security Profile to be created

The following error message appears:

```
"ERROR!  Existing User <username> conflicts with Security Profile to be
created during migration."
```

Problem

The username is the same name as a Security Profile Name that is created by the Metadata Migration process. The Metadata Migration process migrates a 4.0 Group into the following entities:

Example: 4.0 Group name is RM_USERS

Entity Type	Entity Name
User Group	G_RM_USERS
Security Profile	S_RM_USERS
ID Folder	RM_USERS

Because the Group concept in OFSA 4.0 embodied the functionality for User Groups, Security Profiles and ID Folders, the Metadata Migration process creates separate entities from a single Group. The problem identified by the Migrate check is that there is a User or Group Name that is the same name as a Security Profile to be created by the Migrate. In the example, there is a 4.0 Group named RM_USERS and S_RM_USERS.

The following SQL identifies any conflicts that might exist for User Groups to be created by the migrate:

```
SELECT username FROM all_users
      WHERE username IN
        (SELECT 'S_' || DECODE(login_name,'<ALL>','ALL',
                                          '<OFSA>','OFSA',
                                          login_name)
         FROM catalog_of_users
         WHERE type_cd = 2);
```

Solution

Rename or remove the Groups in 4.0 that cause this problem.

User or Group in CATALOG_OF_USERS not uppercase

One of the following error messages appears:

```
"ERROR!  User <username> in CATALOG_OF_USERS not uppercase."

"ERROR!  Group <group_name> in CATALOG_OF_USERS not uppercase."
```

Problem

A User or Group in the CATALOG_OF_USERS table is not in uppercase. The 4.5 FDM database requires that all User and Group Names be in all uppercase.

Use the following SQL to identify invalid Users or Groups

```
SELECT login_name, type_cd FROM catalog_of_users
WHERE login_name <> UPPER(login_name);
```

Solution

Update or remove the invalid Users or Groups. To remove, use the 4.0 System Administration application. To update the Users or Groups to be correct, you need to update the following table/column combinations:

CATALOG_OF_USERS.login_name

CATALOG_OF_GROUPS.group_name

CATALOG_OF_GROUPS.user_name

CATALOG_OF_IDS.group_name

When updating, set the value in the column equal to upper(column_name). In situations where such an update statement creates a duplicate entry, you need to remove the records causing the duplication.

User <username> in HARV_USER not uppercase

The following error message appears:

```
"ERROR!  User <username> in HARV_USER not uppercase."
```

Problem

A User in the HARV_USER table is invalid. Users identified in the HARV_USER table are only used for the 4.0 Campaign Navigator application.

Use the following SQL to identify invalid Users from HARV_USER:

```
SELECT user_name FROM harv_user
WHERE user_name <> UPPER(user_name);
```

Solution

Update the user_name field in the HARV_USER table to uppercase.

<group_name> not a valid User Group

The following error message appears:

```
"ERROR!  <group_name> not a valid User Group."
```

Problem

IDs in CATALOG_OF_IDS are attached to a Group that does not exist. The Group is not identified in the CATALOG_OF_USERS tables as a valid Group.

Use the following SQL to identify invalid Groups

```
SELECT DISTINCT group_name FROM catalog_of_ids
WHERE group_name not in
(SELECT login_name FROM catalog_of_users
WHERE type_cd = 2)
AND group_name <> '<INDIVIDUAL>';
```

Solution

Update the records in CATALOG_OF_IDS attached to the invalid Group. Set the group_name field equal to a valid Group. Or, create a Group in 4.0 System Administration to match the invalid Group name. The entries in CATALOG_OF_ IDS then become valid.

SYSTEM_CODE_VALUES Errors

Alpha values found in numeric columns

The following error message appears:

```
Update cannot proceed, non-numeric values have been found in system_code_
values for rows that contain data for numeric columns.
```

Problem:

The database upgrade procedure has found rows in system_code_values that pertain to numeric columns, which have alphanumeric values.

Solution:

The SQL statement that follows retrieves all rows in system_code_values that have alphanumeric values for numeric columns.

```
Select * FROM system_code_values cv
    WHERE EXISTS (SELECT NULL FROM system_info si
        WHERE si.data_code = 1
        AND ( si.table_name = cv.instrument
        OR cv.instrument = 'ALL')
        AND si.column_name = cv.column_name
        AND si.database_type = 'NUMBER')
    AND TRANSLATE (value, 'a 0123456789','a') IS NOT NULL;
```

You may either delete these rows or modify their values so that they are numeric.

Column_name in SYSTEM_CODE_VALUES not uppercase

The following error message appears:

```
"ERROR!  Column_name in SYSTEM_CODE_VALUES not uppercase."
```

Problem

Records exist in the SYSTEM_CODE_VALUES table with values in the column_name field that are not uppercase. The FDM 4.5 database requires that the column_name values for all records be in uppercase.

The following SQL identifies the records with an invalid column_name value:

```
SELECT instrument,column_name,value
FROM system_code_values
WHERE upper(column_name)<>column_name;
```

Solution

Update the invalid records in SYSTEM_CODE_VALUES.

Instrument values in SYSTEM_CODE_VALUES not uppercase

The following error message appears:

```
"ERROR!  Instrument column in SYSTEM_CODE_VALUES not uppercase."
```

Problem

Records exist in the SYSTEM_CODE_VALUES table with values in the instrument column that are not uppercase. The FDM 4.5 database requires that the instrument column for all records be in uppercase.

The following SQL identifies the records with an invalid instrument value:

```
SELECT instrument,column_name,value
FROM system_code_values
WHERE upper(instrument)<>instrument;
```

Solution

Update the invalid records in SYSTEM_CODE_VALUES.

Duplicate values in SYSTEM_CODE_VALUES

The following error message appears:

```
"ERROR!  Duplicate values in SYSTEM_CODE_VALUES."
```

Problem

Duplicate records exist in SYSTEM_CODE_VALUES for the same table (instrument) and column_name combination.

The following SQL identifies the duplicate records:

```
SELECT UPPER(instrument) instrument,UPPER(column_name) column_name,
TRIM(BOTH FROM value) value
FROM system_code_values
GROUP BY UPPER(instrument),UPPER(column_name),TRIM(BOTH FROM value)
HAVING COUNT(*) > 1
```

Solution

Delete the duplicates from the SYSTEM_CODE_VALUES table.

NULL values in SYSTEM_CODE_VALUES

The following error message appears:

```
"ERROR!  Null values in SYSTEM_CODE_VALUES.column_description"
```

Problem

Null values exist in the column_description field of the SYSTEM_CODE_VALUE table.

The following SQL identifies the records with a null value for column_description:

```
SELECT instrument,column_name,value
FROM system_code_values
WHERE column_description IS NULL;
```

Solution

Update the invalid records in SYSTEM_CODE_VALUES.

SYSTEM_INFO Errors

Duplicate DISPLAY_NAME values in SYSTEM_INFO

One of the following message appears:

```
Update cannot proceed, Tables have been found in system_info whose columns
have the same display_name.
```

```
ERROR!  <table_name>  has columns with duplicate display_name values
```

Problem

Multiple columns of a single table have the same DISPLAY_NAME. The DISPLAY_NAME of columns must be unique within each table.

The following query identifies the rows in SYSTEM_INFO with duplicate DISPLAY_NAME values within a table:

```
SELECT data_code, table_name, column_name, display_name from system_info
WHERE (table_name, display_name)
IN (SELECT  table_name, display_name
      FROM system_info
      WHERE data_code = 1
      AND display_name IS NOT NULL
      GROUP BY table_name, display_name
      HAVING count(*) > 1)
ORDER BY table_name, display_name, column_name;
```

Solution:

Modify the DISPLAY_NAME as appropriate to ensure uniqueness within each table.

Null values found in SYSTEM_INFO columns

The following error message appears:

```
.Update cannot proceed, NULL values have been found in one or more of the
following columns in system_info - display_flag, display_name, and column_
data_type.
```

Problem:

The database upgrade process has found rows in SYSTEM_INFO where either the DISPLAY_FLAG, DISPLAY_NAME or COLUMN_DATA_TYPE are null. Data is required in these columns for OFSA to function properly.

The following query identifies the rows in SYSTEM_INFO with null values for these columns:

```
SELECT data_code, table_name, column_name, display_flag, display_name,
column_data_type
   FROM system_info
   WHERE display_name IS NULL
   OR display_flag IS NULL
   OR column_data_type IS NULL;
```

Solution

You must provide values for these columns before proceeding with the database upgrade process.

The DISPLAY_FLAG column designates whether or not the specific column is displayed in OFSA. *Y* designates that OFSA displays the column, while *N* designates that the column is not displayed.

The DISPLAY_NAME column specifies the name of the column as it appears to the user within OFSA.

The COLUMN_DATA_TYPE designates how the column is used within OFSA (for example, Balance or Rate, and so on).

Use the following matrix to determine the appropriate column_data_type for a column:

Column Data Type	Data Type	Definition
BALANCE	NUMBER(14,2)	Used for columns that hold money values such as a bank account balance
CODE	NUMBER(5)	.Code column with description stored in SYSTEM_CODE_VALUES
CODE_NUM	NUMBER(10)	.Code column with no description in SYSTEM_ CODE_VALUES.
DESCRIPTION	VARCHAR2(255)	A description column.
FLAG	NUMBER(1)	Boolean On / Off flag column (represented by a 1 or a 0).
FREQ	NUMBER(5)	Identifies event frequency (for example, Payment Frequency).
ID	NUMBER(14)	Identifies a leaf column.
IDENTITY	NUMBER(10)	Identifies the source of data.
ID_NUMBER	NUMBER(25)	Used to uniquely identify account records (for instrument / service data).
MULT	CHAR(1)	A *Multiplier* column identifying the unit of measurement for a Term or Frequency. Acceptable values include *D* for Day, *M* for Month, and *Y* for Year.
RATE	NUMBER(8,4)	A percentage value of a monetary balance such as Interest Rate or Rate of Return.
SWITCH	NUMBER(5)	A boolean On / Off flag column (represented by a 1 or a 0).
SYS_ID_NUM	NUMBER(10)	Each value is unique within the entire database. Only used for ID identifier columns reserved for OFSA internal use.
TERM	NUMBER(5)	Duration (used together with MULT)
VARCHAR2	VARCHAR2	Used for any character column that does not fit a described category.

Column Data Type	Data Type	Definition
NUMBER	NUMBER	Used for any numeric column that does not fit a described category.

Tables in SYSTEM_INFO have the same display_name

The following error message appears:

```
"ERROR!  Tables in SYSTEM_INFO have the same display_name."
```

Problem

Each table record in SYSTEM_INFO must have a unique display_name. Table records are designated by data_code>1.

Use the following SQL to identify tables with identical display_name values:

```
SELECT count(*), display_name FROM system_info
WHERE data_code > 1 AND data_code != 4
AND display_flag = 'Y'
GROUP BY display_name
HAVING count(*) > 1;
```

Solution

Update the display_name field in SYSTEM_INFO so that all of the tables (data_code>1) have unique display_name values. No display_name values should be null.

Table, Column Name or Related_Field in SYSTEM_INFO not uppercase

One of the following error messages appears:

```
"ERROR! table_name in SYSTEM_INFO not uppercase."
"ERROR! column_name in SYSTEM_INFO not uppercase.
"ERROR! related_field in SYSTEM_INFO not uppercase."
```

Problem

The metadata in the SYSTEM_INFO table is invalid. All Table Names and Column Names in SYSTEM_INFO must be uppercase values.

Use the following SQL to identify invalid entries in SYSTEM_INFO

```
SELECT table_name,instrument, column_name, data_code, related_field
FROM system_info
WHERE upper(table_name) <> table_name
OR upper(column_name) <> column_name
OR upper(related_field) <> related_field
ORDER BY table_name, column_name
```

Solution

Update the SYSTEM_INFO table to set the values for table_name, column_name and related_field to be uppercase for the invalid records. The related_field column, legal values are either a valid column_name from the same table_name, or N/A.

13

Installing and Configuring Discoverer

This chapter provides information on installing and configuring Oracle Discoverer for use with the Oracle Financial Data Manager (FDM). Complete the steps described in this chapter after creating an FDM database or performing a database upgrade process for an FDM database. The specific topics covered in this chapter include:

- Overview of Discoverer Business Areas
- Installing the End User Layer
- Upgrading Business Areas from Previous OFSA Versions
- Importing OFSA Business Areas for Discoverer
- Market Manager Business Areas and Standard Reports
- Installing and Configuring the OFSA Standard Reports

Additional information on installing and configuring the Oracle database for the Oracle Financial Services Applications (OFSA) group of applications is found in the reference manuals and installation guides associated with the Oracle database and applications you are installing.

Overview of Discoverer Business Areas

Included with the OFSA CD are pre-configured Discoverer Business Areas for:

- OFSA Standard Reports
- Market Manager Reports

These Business Areas enable users to run the seeded reports against the FDM database.

This chapter provides instructions for creating the required Discoverer End User Layer (EUL) as well as for loading the OFSA Business Areas and Market Manager Business Areas into this EUL by importing various export files (EEX) using the Oracle Discoverer 3.1 Administration edition.

Note: FDM 4.5 supports the use of the Market Manager 4.0 business areas and standard reports. However, the Market Manager application and Market Manager Business Areas are not included on the OFSA 4.5 CD. Use the OFSA 4.0 CD for any instructions in this chapter regarding installation of the Market Manager business areas.

For information on how to use the OFSA Standard Reports and Business Areas, refer to the *Oracle Financial Data Manager Reporting Administration Guide*. For information on how to use the Market Manager Reports and Business Areas, refer to the appropriate documentation for Oracle Market Manager (OMM).

Installing the End User Layer

Before using Discoverer, you must install an End User Layer (EUL). An EUL is a set of tables that contains all the information that Discoverer requires to operate. To use the OFSA Business Areas for Discoverer, an End User Layer named OFSA_ EULOWNER is required:

The Discoverer End User Layer for the OFSA Business Areas must be named OFSA_EULOWNER. However, you can install and use additional public EULs if you want.

Complete the following steps to create an Oracle user and public Discoverer 3.1 EUL owned by OFSA_EULOWNER.

1. Log in to Discoverer 3.1 Administration Edition as either the SYSTEM user or the FDM Schema Owner.
2. When asked if you want to create an EUL, select Yes.
3. Select Create an EUL from the options available when the EUL Manager dialog appears.
4. Select Create a new user and make sure that Grant access to PUBLIC is chosen when the Create EUL Wizard appears.

5. Enter OFSA_EULOWNER for the username and a password that you select.
6. Choose Next in the wizard and then select the appropriate Default and Temporary tablespaces for the EUL objects. After making these selections choose Finish in the wizard.
7. Repeat these steps to create additional EUL Owners beyond the required OFSA_EULOWNER End User Layers.

Upgrading Business Areas from Previous OFSA Versions

This section describes how to preserve user customizations of OFSA business areas when upgrading from OFSA 4.0.

Note: The instructions in this section apply only for situations where you are upgrading from OFSA 4.0 and want to preserve customizations of the OFSA business areas. Skip this section and proceed to the Installing the End User Layer if you are new to FDM 4.5 installation.

Oracle Discoverer 3.1 Administration Edition does not support upgrade of OFSA business areas from OFSA 4.0 to FDM 4.5. These business areas were previously stored in OFSA_EULOWNER and OFSA_SYSTEM End User Layers. If you customized the OFSA business areas in OFSA 4.0 and want to preserve those customizations, perform the following procedures to incorporate the contents of those folders into the new OFSA 4.5 business areas.

FDM Reserved Business Area Names

FDM 4.5 reserves the following Oracle Discoverer business area names:

- FDM
- FDMA
- PA
- RM
- RTM

Note: This procedure assumes that none of your existing business areas from OFSA 4.0 conflict with the FDM reserved business area names. If any of your existing business areas conflict with the business area names reserved in FDM 4.5, rename the existing folder to a new name prior to performing any of the procedures.

Migrating Business Areas for OFSA_EULOWNER:

1. Login to the Discoverer Administration Edition as OFSA_EULOWNER. This is typically where you set up your previous user-defined OFSA Business Areas.
2. Select Cancel when the Load Wizard dialog appears.
3. Export the customized business areas that you want to save to a backup location on your PC. To export these business areas, select File>Export, and enter a destination filename. Click Save.
4. Skip the next section for Installing the End User Layer.
5. Follow the instruction described in Importing OFSA Business Areas for Discoverer OFSA.
6. Log into the Discoverer Administration Edition as OFSA_EULOWNER.
7. Select Open an existing business area.
8. Click Select Al and then Finish.
9. Ensure that you have the six new OFSA business areas. If not, repeat step 5.
10. Expand an OFSA business area into that you want to place one of your customized folders.
11. Highlight the user-defined (customized) folders. Hold down the shift key for multiple selection.
12. Drag and drop the desired folders into the new OFSA business area. Note that the folder name must be unique. Rename the folder if necessary using the Folder Properties dialog invoked by right clicking on the folder.
13. Repeat steps 9 through 11 for each OFSA business area into which you want to place customized folders.

Migrating Business Areas for OFSA_SYSTEM:

In FDM 4.5, the OFSA_SYSTEM End User Layer no longer exists. However, the business areas in this EUL are merged into the new FDM business area in OFSA_EULOWNER. If you have modified any of the business areas in the OFSA_SYSTEM EUL and you want to incorporate their contents into the new FDM 4.5 business areas, perform the following procedures:

1. Login to the Discoverer Administration Edition as the EUL owner, that is, OFSA_SYSTEM, which contains the OFSA System Business Areas.
2. Select Cancel when the Load Wizard dialog appears.
3. Export the business areas that you want to save to a backup location on your PC. To export the business areas, select File>Export, and enter the filename you want to save as, for example, my_ofsa_system_ba.eex. Click Save.

> **Note:** The File>Export option in Discoverer Administration is not available unless a Business Areas is highlighted.

4. Select Open an existing business area. Click Select All and then Finish.
5. Expand the new FDM business area. This is the place to which your previous customized folders exported from OFSA_SYSTEM EUL should go.
6. Import the my_ofsa_system_ba.eex file created in step 3. To import the eex file, select File>Import, and navigate to the eex file on your PC. Leave the default Import Options as selected, and then double click on the Business Area filename on your PC. Wait until the business area file has been imported.
7. Highlight the imported business area and select File>Refresh>Finish. This action launches a process that associates the structures within the business area with the correct tables in the OFSA database. Select OK for any dialogs that appear.
8. Expand the customized business area you want to preserve.
9. Highlight the user-defined folders. Hold down the shift key for multiple selection.
10. Drag and drop the desired folders into new FDM business area. Note that the folder name must be unique. Rename the folder if necessary using the Folder Properties dialog invoked by right clicking on the folder.

Note: Refer to the *Oracle Discoverer 3.1 Administration Guide* for more information.

Importing OFSA Business Areas for Discoverer

The OFSA Business Areas are created and configured using the Discoverer Integrator and Discoverer Administration Edition. To run the OFSA Standard Reports, you import these Business Areas into the OFSA_EULOWNER EUL, although you can use other EUL Owners in addition to OFSA_EULOWNER if you want. For names and descriptions of the database tables contained in the OFSA Business Areas, refer to the *Oracle Financial Data Manager Reporting Administration Guide*.

Caution: Before proceeding with this installation, you must either have the FDM database, Release 4.5, installed on your server or have upgraded your FDM database to Release 4.5.

To install the OFSA Business Areas, complete the following steps.

1. Login to FDM Administration as a user with the authority to manage security.
2. Register the OFSA_EULOWNER within FDM Administration using the User Registration wizard.
3. Within FDM Administration, grant the following roles to the OFSA_ EULOWNER user:
   - OFDM_R_REPORT_MART
   - OFDM_R_BUSINESS_PROCESS
   - OFDM_R_FDMA_RPT

   Of course, you may want to grant additional roles to the OFSA_EULOWNER user. The roles listed constitute only the minimum privileges that are required for the OFSA_EULOWNER user.
4. Install the Discoverer End User Layer (EUL) export files on the client. The client must also have Discoverer 3.1 Administration Edition installed.

   When you select the Standard Reports for Oracle Discoverer (within the FDM), the Oracle Installer creates the following directory on the PC:

$ORACLE_HOME/OFSA45/disco31

and copies the following files into that directory:

EEX File Name	Description
fdm_ba.eex	FDM Business Area
fdma_ba.eex	FDM Administration Business Area
pa_ba.eex	Performance Analyzer Business Area
rm_ba.eex	Risk Manager Business Area
rtm_ba.eex	Rate Manager Business Area

5. Log into the Discoverer Administration Edition as OFSA_EULOWNER.
6. Select Cancel when the Load Wizard dialog appears.
7. To import the OFSA Business Areas, select File>Import. Leave the default Import Options as selected, and then double click on the Business Area filename. Do this for each OFSA Business Area file until all of them have been imported.
8. Highlight the business areas and select File>Refresh>Finish. This action launches a process that associates the structures within the business area with the correct tables in the OFSA database. Select OK to any dialogs that appear.
9. Grant access to the appropriate OFSA Business Areas to the FDM Schema Owner and all users that need to use the areas. You must refer to the *Oracle Discoverer 3.1 Administration Guide* for the specific steps required to complete this grant access procedure.

Market Manager Business Areas and Standard Reports

FDM does not include any Market Manager business areas or standard reports for version 4.5. This is because there is no version 4.5 of Market Manager. Rather, FDM 4.5 supports the 4.0 Market Manager application. Any of the 4.0 Market Manager business areas and standard reports can be used in an FDM 4.5 database with Market Manager installed.

Note: If you are upgrading from Market Manager 3.5/4.0, you do not need to re-install the Market Manager Business Areas. These business areas are unchanged in version 4.5. You only need to import the Market Manager Business Areas if you are installing Market Manager for the first time in your database.

If you do need to re-install your 4.0 Market Manager Business Areas and/or Standard Reports, follow the Market Manager specific instructions included in the OFSA Database Installation chapter of the OFSA 4.0 Installation and Configuration Guide.

Installing and Configuring the OFSA Standard Reports

The OFSA Discoverer Standard Workbooks are copied to the PC during the client-side installation process. However, in order to use these workbooks, you must populate the Discoverer EUL using Discoverer Integrator and then map the workbooks to it. For instructions on this process, refer to the *Oracle Financial Data Manager Reporting Administration Guide*.

14

FDM Security

This chapter provides information on the database security framework of the 4.5 Oracle Financial Data Manager (FDM) database. The FDM security framework describes the constructs and tools available to the administrator for managing database and application security for the Financial Data Manager environment. Included in this discussion is information regarding how the security model was migrated from an Oracle Financial Services Applications (OFSA) version 3.5 and 4.0 database.

The information in this chapter is supplemental to the information provided in the *Oracle Financial Data Manager Administration Guide*. The *Oracle Financial Data Manager Administration Guide* provides detailed information about using the FDM Administration application, including information about the various security features available for the FDM database.

The following, specific topics are covered in this chapter:

- FDM Schema Owner
- Database and Application Privileges
- FDM Security Framework
- Division of Administrative Responsibilities
- Managing Security for the Reporting Data Mart
- Troubleshooting Privilege Errors
- Migration from Version 3.5 or 4.0 Security

FDM Schema Owner

The FDM Schema Owner is the Oracle RDBMS username that owns all of the FDM Reserved database objects. This username is the source for all privileges within the FDM database. By definition, the FDM Schema Owner is an all powerful user account with no security restrictions.

The FDM Schema Owner is identified during the FDM Database Creation Process. This process creates an entry in the OFSA_DB_INFO table identifying the owner of the FDM Reserved objects. In order for a database to be valid, the entry in OFSA_DB_INFO must be the same user that owns the FDM Reserved objects.

The FDM Schema Owner is the only user account that is allowed to grant administration privileges to other users within the FDM Administration application.

Database and Application Privileges

This section describes how to implement security for the Financial Data Manager database environment. The following specific topics are discussed in this section:

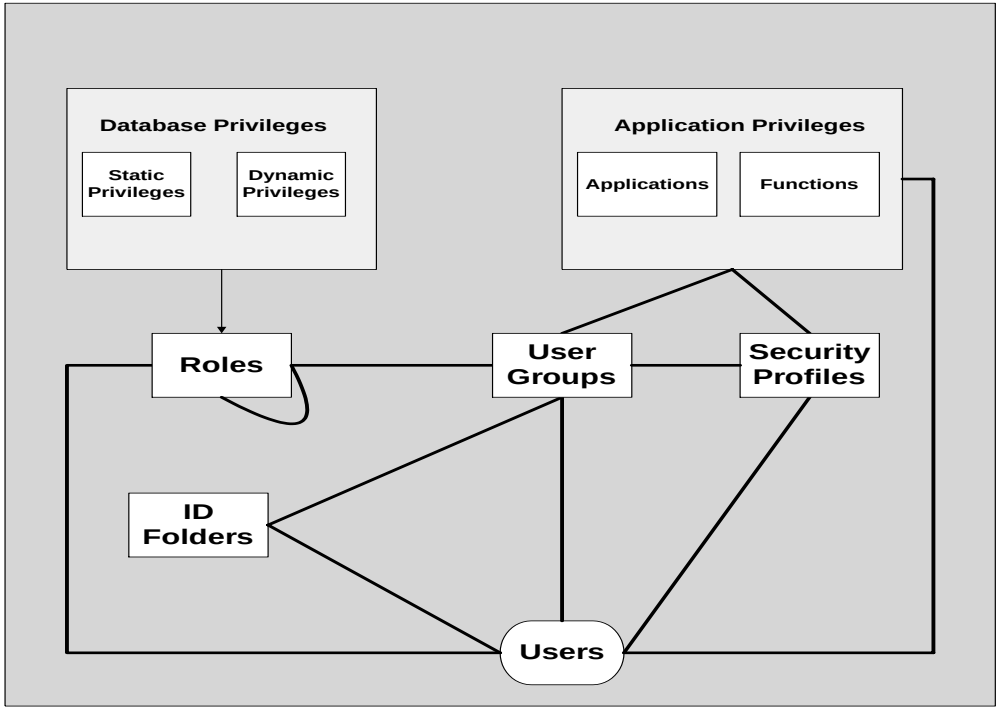
- FDM Security Framework
- Universal Login
- Database Object Privileges
- Roles
- Assigning and Revoking Database Privileges
- Oracle Password Aging, Expiration and History
- FDM Grant Procedures
- Supporting Seeded Data

Note: This section discusses FDM database privileges in detail. OFS Application privileges (which includes both Application and Function privileges) are discussed more completely in the *Oracle Financial Data Manager Administration Guide.*

FDM Security Framework

The FDM Security Framework describes how security is implemented within the FDM database environment for the OFS applications. The FDM Security Framework is composed of different entities and types of privileges used for implementing security.

By default, registered users have no privileges unless specifically granted by the administrator. However, exceptions to this include two seeded roles (OFDM_R_LOGIN and OFDM_R_ID) granted to users automatically by the FDM Administration application during user registration.



The various entities and privilege types employed in the FDM Security Framework are described as follows:

Database Privileges

Database Privileges are defined as the ability to perform SELECT, INSERT, UPDATE, DELETE or EXECUTE operations against database object. Database Privileges are categorized as either Static or Dynamic:

- Static Privileges are on FDM objects that always exist
- Dynamic Privileges are on FDM objects created by Risk Manager and Transformation processing

Application Privileges

Application Privileges are defined as the ability to use OFS applications.

- Application Privileges are the ability to login to an OFS application
- Function Privileges are the ability to perform specific operations within an OFS application

Roles

Roles are collections of database object and database system privileges. Assign roles to User Groups so that all members within the User Group receive the role. You can also assign Roles directly to individual users.

Security Profiles

Security Profiles are collections of application privileges. Assign Security Profiles to User Groups so that all members of the User Groups receive the associated privileges. You can also assign Security Profiles directly to individual Users.

User Groups

User Groups are collections of Users. Assign Roles and Security Profiles to User Groups so that all members within the User Group receive the associated privileges.

ID Folders

ID Folders are collections of OFSA IDs. Assign ID Folders to User Groups so that all members of the User Group have access to the ID Folder. You can also assign ID Folders directly to individual users.

Users

Users are registered FDM Users.

Universal Login

The concept of a Universal Database Login embodies the ability to login to the FDM database from any application or SQL compatible tool using a single login account. FDM 4.5 supports this concept by distinguishing database privileges for Internal use or External use. Internal privileges are available to users only when logged into one of the OFS applications. External privileges are always available to users.

Users therefore only require a single user account for the FDM database. This is in contrast to previous versions of OFSA, in which users needed two users accounts - one for the OFS application operations and the other for reporting.

Database Object Privileges

Privileges for FDM Reserved Objects

FDM Reserved Objects are those objects created by the FDM database installation that are required for OFS application operations. FDM Reserved objects include all of the database object names reserved by the applications, such as table names, sequence names, view names, etc. FDM Reserved names are prefixed with OFSA_ for easy identification.

In general, privileges for FDM Reserved Objects are provided by seeded roles. Seeded roles are created during the FDM database upgrade and database creation processes. Each role is associated with a particular OFS application and provides all of the access required for operations within that application.

The FDM database includes seeded Internal roles for operations performed when logged into one of the OFS applications. FDM also provides seeded External roles for user reporting and querying.

Privileges for Client Data Objects

Client Data Objects include any objects not created by the initial installation of the FDM database. This category of objects is comprised of objects created by administrators as well as the LEDGER_STAT table. LEDGER_STAT is technically a reserved table name because FDM reserves this particular table name as well as a particular table structure for it. However, because LEDGER_STAT stores client account and statistical data, privileges for this table are the purview of the administrator and are not included in the seeded roles.

It is the administrator's job to create roles to provide privileges on Client Data Objects to the users. By default, users have no database privileges on these tables,

except for users that migrated from a 3.5 or 4.0 database. All users require access to these objects in order to perform their duties.

Privileges for Dynamic Objects

Dynamic objects are those tables created by Risk Manager and Transformation output processing. These tables are considered to be Client Data Objects, and are therefore managed by roles created by the administrator. However, because these objects are created during processing rather than directly by the administrator, security management is for them different than that for static objects.

The FDM Administration application provides functionality for managing privileges for dynamic objects. Refer to the *Oracle Financial Data Manager Administration Guide* for more information on how to manage privileges for dynamic objects.

Roles

The FDM 4.5 database security framework relies on roles to provide database privileges for users. Administrators are allowed to create their own roles to manage database privileges, as well as use any existing roles that they already have. This flexibility facilitates the integration of FDM with other data stores.

Internal and External Roles

Roles registered in FDM are categorized as either Internal or External.

Privileges in External roles are available to users during any database session. External roles provide privileges that are meant to be available to users at all times, within any database session.

Internal roles are password protected roles that are available to the users only when they are logged into one of the OFS applications. The OFS applications automatically enable (via the set role command) any registered Internal roles assigned to the user within the FDM metadata during login. Only roles registered with FDM and assigned to the user using the FDM Administration application are enabled in this manner.

Internal roles provide privileges that are meant to be available to users only when logged into one of the OFS applications. In general, users require greater privileges for operations and processes within the individual OFS applications. Because Internal roles are password protected, they provide a mechanism to grant such privileges to users that are not available in unprotected database sessions, such as SQL*Plus.

Role Registration

Any role that the administrator intends to use with the FDM database should be registered using the FDM Administration application. Role registration provides several benefits for the administrator, including:

- the ability to distinguish privileges available only within the applications (Internal Roles) versus privileges available in any database session (External Roles).
- automatic refresh of any privileges assigned within the FDM Administration application by way of the Grant All procedure.
- the ability to assign Roles to FDM User Groups as well as users.
- the ability to associate Table Classifications and Table Names for dynamic privileges

For more information regarding Role Registration, refer to the *Oracle Financial Data Manager Administration Guide.*

Caution: The FDM Role Registration process removes Internal roles from the default list for all registered FDM users. These roles are not available for FDM user's database session unless they are logged into an OFS application. Register roles as Internal only if you intend for the database privileges conveyed by the role to be exclusively available within OFS applications.

Sharing Roles within a Data Store

Any role that exists in the database instance can be registered with the FDM metadata, including roles used by other applications. Roles shared with other applications should always be registered as External roles because they provide privileges that need to be available within any database session. Registering such roles enables the administrator to minimize the number of roles required to manage security within the instance.

Role Passwords

FDM requires that all Internal roles are password protected. This ensures that privileges for Internal roles are available to users only when they are logged into one of the OFS applications.

The Oracle password for the role is not encrypted, however, the Role passwords are stored in an encrypted form in the FDM database to prevent users from discovering role passwords by querying the FDM Metadata. The OFS applications read this encrypted form of the password during the role enabling process. Role passwords are encrypted using a key value from the OFSA_CONTROL table. The FDM database creation and upgrade processes seed OFSA_CONTROL with a default encryption key.

Only the FDM Schema Owner has privileges to the OFSA_CONTROL table. Oracle recommends that you do not allow any other users access to this table. There is no reason to change the encryption key value unless your role passwords have been compromised. If you do change the value of your encryption key in OFSA_CONTROL, users are unable to access any Internal roles until you reset the passwords in FDM Administration. To do this, login to the FDM Administration application and click the Change Password button for each password protected role. FDM Administration then uses the new value in OFSA_CONTROL to encrypt the role password.

Note: FDM encrypts the role password only for storing it in the FDM Metadata. When you specify the role password, either in FDM Administration or directly in SQL*Plus, that password is the real password for the role. This enables you to share password protected roles with other applications in the database instance because the *real* role password is not encrypted.

Caution: All passwords for FDM seeded Internal roles are set to XXADCFGZ by the FDM database upgrade and database creation processes. Oracle recommends that you change these role passwords by using the Password button on the Roles tab within the FDM Administration application.

Assigning and Revoking Database Privileges

The rules for assigning and revoking of database privileges within FDM Administration require explanation due to the nature of these operations:

Assigning Database Privileges

When you grant an database object privilege or a role within the FDM Administration application, the application automatically executes a grant statement for that privilege within the Oracle RDBMS. Thus, if you assign a role to a User Group within FDM Administration, the role is immediately granted directly within the Oracle RDBMS to all users of that User Group. If you assign the SELECT privilege on the DEPOSITS table to a role using the FDM Administration application, the application immediately executes a grant statement for the object privilege as well. This is true for both Internal and External roles for any privileges or role assignments performed within FDM Administration.

Assignment of Dynamic privileges, however, does not result in any immediate grant statement. When you make a dynamic privilege assignment within the FDM Administration application, the privilege assignment is operational for any future dynamic objects created by the OFS applications. For example, if you assign the SELECT privilege to a role on the RM Detail Results Table Classification, then that role receives that privilege for any future objects of that Table Classification. It does not receive privileges on existing RM Detail Results tables, unless you run the Grant All or Grant All Dynamic Privilege procedures.

Revoking Privileges

Database Object and System privileges are immediately revoked from a role if you remove the privilege within FDM Administration. FDM Administration automatically executes the revoke statement for the privilege within the Oracle RDBMS.

However, if you remove a role from a recipient with the FDM Administration application, the application only revokes the role from the user if it is an Internal role, and if it is the only remaining source from which the user receives the role. For example, assume that a user belongs to a User Group that is currently assigned the OFDM_R_RM role. If you remove the role from the User Group, FDM Administration only revokes the role from the user within the Oracle RDBMS if the user does not receive the role from another source, either directly, or from another User Group of which it is a member. In this example, if the user also was a member of another User Group that also is assigned the OFDM_R_RM role, then removing the role from the (first) User Group does not result in a revoke statement. The user continues to receive the privilege from the role.

Unregistering a User revokes all FDM registered Internal roles from that user. Unregistering a User Group revokes all FDM registered Internal roles assigned to that User Group from the User Group members if that User Group was the only source from which they received the role(s).

External roles are never revoked from users within FDM Administration. Because these roles are potentially shared within a data warehouse, the FDM Administration application never issues any revoke statements for External roles. To revoke these roles from a User, you must remove any assignments of them from the user within FDM Administration and then manually issue the revoke statement within the Oracle RDBMS.

Note: External roles are never revoked within FDM Administration. To revoke External roles from a user, remove the assignments within FDM Administration and then manually issue the revoke statement within the Oracle RDBMS.

Removing a Dynamic Privilege assignment within FDM Administration only removes the privilege for future dynamic tables created by the OFS applications. It does not revoke the privilege from the user on any existing objects. To revoke privileges on existing dynamic objects, remove the role receiving the dynamic privilege from the user and then manually issue revoke statements for any existing objects from that user.

Oracle Password Aging, Expiration and History

The Oracle8i RDBMS provides functionality for password aging, expiration, and history. These features can be implemented within the FDM 4.5 database environment. However, Oracle error messages generated as a result of these features do not display the message text properly for any of the client OFS Applications that rely upon the 16 bit SQL*Net technology stack for database connectivity. If you have implemented any of the password aging, expiration, and history features native to the Oracle 8i RDBMS, your users receive message-not-found errors in these applications whenever their password is expired or their account is locked.

Examples of such message are:

```
Message 28000 not found
Message 28001 not found
```

The actual message text for these errors is:

```
ORA-28000 account is locked
ORA-28001 the password has expired
```

Note: If you have implemented any of the password expiration features, the database may require the users to specify a new password at login. However, the OFS applications do not allow users to enter a new password at the login prompt. In order to enter a new password, users must login to SQL*Plus directly. SQL*Plus then prompts the user to enter a new password if their current password is expired. This behavior applies to all of the OFS applications, not just the ones relying upon the 16 bit SQL*Net.

The OFS client applications relying upon the 16 bit SQL*Net technology stack are:

- Oracle Balance and Control
- Oracle Performance Analyzer
- Oracle Risk Manager
- Oracle Transfer Pricing

All other OFS applications (Oracle Discoverer Integrator, FDM Administration, and Oracle Rate Manager) display the appropriate message text for password aging, expiration, and history errors.

Caution: Do not display the prompt for new passwords.

FDM Grant Procedures

FDM provides several procedures for refreshing and reapplying privileges. These procedures are all part of the Grant All process. Launching Grant All executes each of the following individual procedures:

- Grant All Object Privileges (grant_all_db_obj_privs)
- Grant All Roles (grant_all_db_roles)
- Grant All Dynamic Privileges (grant_all_dyn_obj_privs)
- Analyze All Objects (analyze_all)
- Create Public Synonyms (create_all_public_synonyms)

Each of these tasks can be launched individually or as a collective unit from within the FDM Administration application.

> **Note:** Analyze All is only available as a separate procedure from within the FDM Administration application. Because Analyze All may require significant time to complete when run on a database with a large number of rows, this procedure is not executed automatically when you launch Grant All from within the FDM Administration application. However, Analyze All is executed automatically when you run Grant All from the command line in SQL*Plus.

To launch all of the procedures as a collective unit, run the Grant All menu option from within FDM Administration or execute the following directly in SQL*Plus:

```
SQL> execute ofsa_dba.grant_all(login_name, execution_mode);
```

The login_name parameter passed to the procedure identifies how log entries for the process are stored in the audit table (OFSA_STP). Pass in the login_name parameter within single-quote marks. The execution_mode parameter designates that you are running the procedure from the command line in SQL*Plus. When running Grant All from SQL*Plus, pass this parameter as COMMAND_LINE. For example:

```
SQL> execute ofsa_dba.grant_all('OFSA', 'COMMAND_LINE');
```

To launch the procedures individually, run the appropriate menu option within the FDM Administration application, or execute the following directly in SQL*Plus:

```
SQL> execute ofsa_dba.procedure_name(login_name);
```

The procedure_name is one of the individual procedure names in parenthesis. The execution_mode parameter is not required to launch the procedures individually.

Revoke of Unassigned Internal Roles by Grant All

In addition to executing each of the following procedures, Grant All also revokes any unassigned Internal roles from FDM user. An unassigned Internal role is a role registered as Internal to which the user does not have an assignment in the FDM Metadata. Grant All compares the role assignments in the FDM Metadata to the current role privileges in the Oracle RDBMS catalog. Any Internal roles assigned to users in the Oracle RDBMS catalog that are not assigned to users in the FDM Metadata are immediately revoked.

This revoke ensures that users receive Internal role privileges only when logged into an OFS application. Do not grant roles registered as Internal in FDM to users outside of the FDM Administration application (in other words, do not grant such roles to users in SQL*Plus). Such grants invalidate the FDM security implementation. If your users require privileges external to the OFS applications, grant External roles to the users within FDM Administration.

Grant All Object Privileges

This procedure re-applies any grants on static objects assigned using the FDM Administration application. For example, if the SELECT privilege on the DEPOSITS table is assigned to a role using FDM Administration, this procedure re-executes the grant statement for that privilege. This is useful for situations where existing grants are no longer valid.

Grant All Roles

This procedure refreshes grants of roles to users assigned using the FDM Administration application. Roles granted to User Groups are also refreshed for each of the individual members of the User Group.

This procedure also revokes any FDM registered Internal roles that are not assigned to the user within the FDM Metadata but are still designated as granted to that user within the Oracle RDBMS catalog.

Grant All Dynamic Privileges

This procedure refreshes any grants on dynamic objects assigned using the FDM Administration application. For roles associated with Table Classifications (Dynamic Table Classification Privileges within FDM Administration), this procedure re-executes the privileges assigned to the roles for all objects of the designated Table Classification. For example, if the administrator assigns the SELECT privilege to the RM_REPORTING role for the Risk Manager Results Table Classification, this procedure re-grants this privilege to the RM_REPORTING role for all objects of that classification.

For more information regarding how to assign dynamic privileges, refer to the *Oracle Financial Data Manager Administration Guide.*

Analyze All Objects

This procedure performs the analyze table estimate statistics SQL command for all tables registered within the FDM metadata.

Create Public Synonyms

This procedure drops and recreates public synonyms for all registered objects within the FDM metadata.

Troubleshooting FDM Grants Procedures

The Grant All procedure (and any of its component procedures) records a log entry into the OFSA_STP table for all DDL statements that it executes. This includes grant statements (such as grant select on object_name to username) as well as any object creation statements (such as public synonym creation). These log entries are recorded with an audit identifier labeled as the USERNAME column in the OFSA_STP table.

If you are encountering issues with any of the grants procedures, review the entries in the OFSA_STP table for additional information. Oracle recommends that you truncate the OFSA_STP table prior to attempting to review any log entries, to make it easier to find the information pertaining to your particular run. Because the data stored in the OFSA_STP table is only log data, you can truncate this table at any time.

Common Errors

One error that does tend to occur when databases are imported and exported within the same instance is the following (in this example PKMOFSA is the name of the FDM Schema Owner):

```
ERROR at line 1:
ORA-04021: timeout occurred while waiting to lock object PKMOFSA.HROLE
ORA-06512: at "PKMOFSA.HROLE", line 568
ORA-06512: at "PKMOFSA.HROLE", line 595
ORA-06512: at "PKMOFSA.OFSA_DBA", line 88
ORA-06512: at line 1
```

This error occurs because one or more of the Market Manager version 4.0 roles (QUERY, USER or ADMIN) was originally created by a username other than the current FDM Schema Owner. The Grant All procedure is attempting to drop these roles, and is unable to lock the object. To resolve this problem, manually drop the Market Manager roles and then execute the Grant All procedure again. Because Grant All recreates the roles anyway, manually dropping these roles does not cause any problems.

Supporting Seeded Data

FDM provides seeded data to facilitate assigning database and application privileges to users. This seeded data is comprised of Roles, User Groups and Security Profiles. The seeded data makes it easier for you, as an administrator, to provide users with the privileges that they need to perform required operations within the OFS applications.

FDM Administration seeded data is categorized as either Protected or Unprotected. Seeded entities identified as Protected can be assigned / revoked to / from users, but is restricted from edit or modification. Seeded entities identified as Unprotected are not restricted from edit or modification.

The following seeded data is discussed in this section:

- Roles
- Security Profiles
- User Groups

Roles

Seeded roles providing privileges on FDM Reserved Objects are designated as Protected. Because the nature of FDM Reserved objects are subject to change between releases, roles providing privileges for these objects cannot be edited. Of course, FDM Administration provides the capability for you to create and define your own roles for providing privileges on FDM Reserved Objects. However, in most cases this is unnecessary as the seeded roles provide the appropriate level of security for OFS application operations.

Users are assigned two required roles automatically by the FDM Administration application during User Registration. These roles are OFDM_R_LOGIN and OFDM_R_ID. Both of these roles are assigned directly to the user. They comprise the minimum privileges users need for OFS application operations. These roles cannot be revoked from the user. To revoke the privileges provided by these roles from a user, you must unregister the user from FDM.

FDM also provides seeded roles for privileges on seeded Client Data Objects. The FDM Financial Data Model includes many seeded objects that are not FDM Reserved. These objects include instrument tables, transaction tables and other supporting account data tables. FDM provides seeded roles for these objects that are Unprotected. Administrators are encouraged to edit and modify these roles as necessary, or to create new roles, to provide appropriate database privileges for users.

Users require Internal privileges for performing operations while logged into the OFS applications, such as running processes and defining business assumptions. Users require External privileges for performing reporting and ad-hoc querying. FDM provides seeded roles for both of these privilege categories.

Internal Privileges

To perform Internal OFS application operations, users require privileges on FDM Reserved objects as well as the Client Data objects. In general, users require full privileges (SELECT, INSERT, UPDATE and DELETE) on these objects for internal application tasks. As these privileges are provided to users with Internal roles, the assigned OFS application and functional security privileges control user operations.

In order to simplify security management, each OFS application has a corresponding Internal role for the privileges required to perform operations for that application. In addition, FDM provides a separate role for Internal privileges on the seeded client data tables (such as Instrument tables, Transactions tables, Ledger_Stat, etc.).

The 4.5 Database Upgrade / Creation Process provides the following seeded roles for OFS application operations:

- OFDM_R_BC (Balance and Control)
- OFDM_R_CLIENT_PROC (instrument and Client Data objects)
- OFDM_R_DI (Discoverer Integrator)
- OFDM_R_FDMA (FDM Administration)
- OFDM_R_PA (Performance Analyzer)
- OFDM_R_RM (Risk Manager)
- OFDM_R_RTM (Rate Manager)
- OFDM_R_TP (Transfer Pricing)

Each of these roles provides the privileges needed to perform tasks within the OFS Applications. All of these roles are identified as Internal, except for the OFDM_R_DI role that is an external role (because Discoverer Integrator requires interaction with Oracle Discoverer, this role is identified as an External role). All of these roles, except for the OFDM_R_CLIENT_PROC role, are protected and cannot be modified.

The OFDM_R_CLIENT_PROC role is not specific to any application. Rather, it provides SELECT, INSERT, UPDATE and DELETE privileges on instrument and Client Data objects. For a new installation, it provides these privileges on the instrument, transaction and other account data tables seeded with the FDM

database. For an upgrade from OFSA version 3.5 or 4.0, it provides these privileges on all identified Client Data objects. See the Migration from Version 3.5 or 4.0 Security. The OFDM_R_CLIENT_PROC role is Unprotected.

External Privileges

External privileges are required for reporting and ad-hoc querying operations. Users require such privileges for performing operations in Oracle Discoverer or SQL*Plus. Users also require such privileges for performing queries within the SQL Talk interface available within some of the OFS applications.

The majority of users require the ability to SELECT on objects in the FDM database. For these users, FDM seeds the following External, Unprotected roles:

- OFDM_R_REPORT_MART
- OFDM_R_BUSINESS_PROCESS

These roles are composed of several sub-roles.

> **Note:** In most cases, you need only to assign either or both of the OFDM_R_REPORT_MART or OFDM_R_BUSINESS_PROCESS roles to your users to provide the External privileges that they need for reporting operations. To do this, assign the user to the OFDM_G_REPORT_MART User Group.

A small number of users require the ability to SELECT, INSERT, UPDATE and DELETE on Client Data objects in external database session. For these users, FDM seeds the following External, Unprotected role: OFDM_R_CLIENT_EXT

Those users in charge of Security and Object Management within FDM Administration require the SELECT privileges on the FDM Metadata objects. For these users, FDM seeds the following External, Protected role:

- OFDM_R_FDMA_RPT

> **Note:** Few, if any users, should ever require more than the SELECT privilege on FDM Reserved Objects for External operations. If for some reason you do need to provide such privileges, you need to create your own External roles within FDM Administration.

OFDM_R_REPORT_MART (External/Unprotected)

The OFDM_R_REPORT_MART role provides complete privileges for results reporting (that is., non- Business Process reporting). Assigning this role to a user (or User Group) provides that user with SELECT access on all client data tables, as well as any results tables, code description tables, audit tables and error message tables.

This high-level role is comprised of several sub-level roles. These sub-level roles are:

- OFDM_R_GENERAL_RPT
- OFDM_R_RM_RPT
- OFDM_R_RTM_RPT
- OFDM_R_CLIENT_RPT

OFDM_R_GENERAL_RPT (External/Protected)

This role provides privileges on objects that are not specific to any OFS applications. These are shared objects, or objects that provide functionality for the Reporting Data Mart.

The OFDM_R_GENERAL_RPT role provides SELECT privileges on the following categories of objects:

- FDM Reserved Code Description Tables and Views
- Functions (OFSA_RDM_YTD, other related tables)
- Dynamic privileges on Hierarchies (transformation output tables for Tree Rollups)
- FDM Reserved Error tables (such as OFSA_PROCESS_ERRORS, OFSA_ MESSAGE_LOG)

OFDM_R_RM_RPT (External/Protected)

This role provides SELECT privileges necessary for Risk Manager operations. This includes:

- RM Audit tables (OFSA_PROCESS_CASH_FLOWS)
- RM Static results (OFSA_RESULT_MASTER, VAR objects, Result Bucket and Result Header tables)
- RM Dynamic results (Result Detail tables, EAR results tables)
- Exchange Rate and Interest Rate Audit tables
- Other RM ID and Reserved tables (for Business Process reporting)

OFDM_R_RTM_RPT (External/Protected)

This role provides SELECT privileges for Rates and Conversion tables included in Rate Manager. This role is also used by the OFDM_R_BUSINESS_PROCESS role.

OFDM_R_CLIENT_RPT (External/Unprotected)

This role provides SELECT privileges on seeded client data tables. This includes:

- Instrument
- Transaction
- Ledger_Stat
- Services tables
- Dynamic Privileges for transformed Ledger_Stat

For databases upgraded from OFSA version 3.5 or 4.0, this role is also seeded with SELECT privileges on any Client Data objects.

> **Note:** The OFDM_R_CLIENT_RPT role is designed to be edited by the administrator to change or modify privileges as needed.

OFDM_R_BUSINESS_PROCESS (External/Unprotected)

This role provides privileges necessary to perform Business Processing reporting, such as reporting on the OFSA IDs and Constructs used in the business process. This includes privileges on FDM Reserved tables such as 'OFSA_IDT_DATA_FILTER' or 'OFSA_CATALOG_OF_IDS'.

This role is comprised of individual, application roles. These roles are identical to the Internal roles seeded for each application, except that they provide 'SELECT' access instead of SELECT, INSERT, UPDATE, DELETE. These roles are:

- OFDM_R_BC_RPT
- OFDM_R_PA_RPT
- OFDM_R_RM_RPT
- OFDM_R_RTM_RPT
- OFDM_R_TP_RPT

All of these External application roles are designated as Protected.

OFDM_R_CLIENT_EXT (External / Unprotected)

This role provides SELECT, INSERT, UPDATE and DELETE privileges on instrument and Client Data objects. The 4.5 Database Upgrade Process grants this role to any users that had update privileges within SQL Talk in the 3.5 or 4.0 versions. For a new installation, this role has privileges only on seeded instrument and transactions tables (the same tables as for the OFDM_R_CLIENT_RPT role).

> **Note:** The OFDM_R_CLIENT_EXT role is designed to be edited by the administrator to change or modify privileges as needed.

OFDM_R_FDMA_RPT (External / Protected)

This role provides SELECT access on the FDM Metadata tables for Security and Object Management reporting. Users that are members of the OFDM_G_FDMA_SEC and OFDM_G_FDMA_OBJ User Groups automatically receive this role so that they have the required privileges for running the standard Oracle Discoverer reports for the FDM Administration Business Area.

Security Profiles

Security Profiles are much like Roles in that they are collections of privileges. However, Security Profiles are collections of application privileges specific to the OFS applications, rather than collections of database object privileges. Security Profiles provide users with the ability to access OFS applications, as well as perform tasks and operations (identified as Functions) within them.

FDM provides a seeded Security Profile for each OFS application, except for Discoverer Integrator, which relies on security from Oracle Discoverer. Market Manager is also excluded from this.

The following Security Profiles are seeded with FDM 4.5. These Security Profiles are all designated as Protected and provide full application and function privileges for each corresponding OFS application:

- OFDM_S_BC (Balance and Control)
- OFDM_S_FDMA (FDM Administration)
- OFDM_S_PA (Performance Analyzer)
- OFDM_S_PFA (Portfolio Analyzer)
- OFDM_S_RM (Risk Manager)
- OFDM_S_RTM (Rate Manager)

- OFDM_S_TP (Transfer Pricing)

If you want to provide limited access to OFS application functions, you need to create your own Security Profiles. For more information regarding Security Profiles, refer to the *Oracle Financial Data Manager Administration Guide*.

User Groups

User Groups in FDM are collections of users. User Groups facilitate security management within FDM by enabling you to associate Roles (collections of database privileges) and Security Profiles (collections of application privileges) with User Groups.

FDM provides a seeded User Group for each OFS application as well as a separate User Group for Reporting Data Mart access. These seeded User Groups are:

- OFDM_G_BC (Balance and Control)
- OFDM_G_DI (Discoverer Integrator)
- OFDM_G_FDMA (FDM Administration)
- OFDM_G_PA (Performance Analyzer)
- OFDM_G_PFA (Portfolio Analyzer)
- OFDM_G_REPORT_MART (Reporting Data Mart access)
- OFDM_G_RM (Risk Manager)
- OFDM_G_RTM (Rate Manager)
- OFDM_G_TP (Transfer Pricing)

Each seeded User Group is associated with corresponding seeded application roles (both Internal and External) as well as a seeded application Security Profile. By assigning a User to a seeded User Group, you are providing that User with all of the privileges that they need to perform application operations.

For more information on the seeded User Groups, refer to the *Oracle Financial Data Manager Administration Guide.*

Division of Administrative Responsibilities

The FDM Schema Owner possesses all privileges required for administering the FDM database environment. The FDM Schema Owner can also grant to other users the privileges required to perform security management and object management for the FDM database. This is accomplished by assigning such users to the OFDM_R_

FDMA_SEC and OFDM_R_FDMA_OBJ User Groups within the 4.5 FDM Administration application. These users then have the ability to login to the FDM Administration application and perform security management and object management tasks.

However, users assigned to one or both of these User Groups possess the required privileges for security management and object management only when logged into the 4.5 FDM Administration application. If you require that these users have the ability to perform administrative tasks outside of the FDM Administration application, grant the appropriate Oracle RDBMS privileges to these users via an External role.

Managing Security for the Reporting Data Mart

The Reporting Data Mart is the set of objects within FDM accessed for reporting purposes. Generally, users conduct reporting operations within FDM using tools such as Oracle Discoverer, Oracle Reports, or even SQL*Plus. Use of these tools requires External privileges for the users, meaning that you should assign External roles to the users so that they have their privileges whenever logged into the FDM database.

Within the Reporting Data Mart, there are both Static and Dynamic objects. Static objects are those tables and views that exist permanently (or at least are not dropped and recreated periodically) within the FDM database. Dynamic objects are those that are created by OFS Risk Manager and Transformation processing. Dynamic objects tend to be dropped and re-created periodically by OFS processing.

In order to provide privileges on Dynamic objects, you must assign a Dynamic Object privilege to a Role (or directly to a user) within FDM Administration. Dynamic object privileges consist of assigning a privilege (such as SELECT) on a Table Classification (such as RM Results tables) to a Role or User. When Risk Manager or Transformation processing creates a table of the specified Table Classification, the Role or User receives the designated privilege. FDM Administration also enables you to assign a dynamic privilege for a specific table name, so that when Risk Manager or Transformation processing creates a table of that name, the user or role automatically receives the privilege just for that table, rather than for all tables within a classification.

Generally, users only require the SELECT (External) privilege on objects in the Reporting Data Mart. For any reporting privilege, whether static or dynamic, always grant using External roles.

Refer to the *Oracle Financial Data Manager Administration Guide* for more information about how to assign Dynamic Object privileges.

Troubleshooting Privilege Errors

Users may encounter privilege errors in performing operations within the OFS application. This section describes some of the possible errors.

The FDM Grants procedure (described in this chapter) automatically refreshes all privilege grants and synonyms. Running this procedure ensures that all of the grants are up to date and synchronized with the FDM Metadata and may resolve any Oracle errors that you or your users are experiencing. Oracle recommends that you run this procedure on a regular basis.

> **Note:** When troubleshooting privilege errors, run the FDM Grants procedure to ensure that all privilege grants are up to date and synchronized with the FDM Metadata. Also, use the FDM Administration Standard Discoverer Reports to identify user privileges and the source for those privileges.

Login Errors

Users may receive one of the following errors during login:

- You have no authorization to access applications
- ORA-00942: Table or View does not exist
- ORA-01031: Insufficient Privileges

At a minimum, in order for a user to login to an OFS application, they must be a registered user for FDM with the following privileges:

- OFDM_R_LOGIN Role - FDM Administration assigns this role directly to the user automatically during the User Registration process
- Application Assignment - the user must receive an application assignment either directly to the user, or from a User Group or Security Profile
- Application role - the user must be assigned to the role for the application, either directly to the user or from a User Group.

Java Class and GenAuthKey errors

If you receive any errors referring to invalid Java Classes or GenAuthKey not found, then one or both of the following is true:

- The initjvm.sql package was not loaded successfully into the database
- The FDM jar files were not loaded successfully into the database

To resolve this problem, verify that you loaded the initjvm.sql package properly into your FDM database. Refer to the instructions in Chapter 10, "FDM Database Installation" for details on how to load this package properly.

After you have verified that the initjvm.sql package is successfully loaded into your database, verify that the FDM jar files were loaded properly. Refer to the Chapter 21, "FDM Utilities" chapter for details on how to reload these packages.

Errors During OFS Operations

Users may encounter Oracle database errors relating to object access in performing OFS application operations. Users encounter one of the following two errors when they do not possess the required database privileges for a specified object:

- ORA-00942: Table or View does not exist
- ORA-01031: Insufficient Privileges

The OFS applications and processing operations display objects based upon the FDM Metadata. The object lists presented to users are not restricted to only those objects on which a user has privileges. Rather, the OFS applications display objects based upon the information in the FDM Metadata. It is therefore possible for users to attempt an operation against on object on which they do not possess privileges.

> **Note:** The pick lists and lists of objects presented to users within the OFS applications are not restricted to only those objects on which a user has privileges. These object lists are populated based upon the FDM Metadata and are not specific for each user.

If the user does require access for an object on which they do not possess the appropriate privileges, grant a role with the required privileges either directly to the user, or to a User Group of which they are a member. For OFS application objects (objects named with the OFSA_ prefix), use the appropriate seeded role. For a client data object, either use one of the seeded client data roles, or create and register a new role with the appropriate privilege.

Refer to Supporting Seeded Data for details on the various seeded roles provided in FDM.

Migration from Version 3.5 or 4.0 Security

Security in OFSA versions 3.5 or 4.0 was managed significantly different than the new FDM 4.5 Security framework. Users migrated from a 3.5 or 4.0 database to 4.5 retain the same privileges, both database and application. However, because the OFSA 3.5 and 4.0 versions operated upon a *union of restrictions* paradigm rather than *a union of privileges* paradigm, a migration from these versions creates challenges for ongoing security management. This section discusses how these users are migrated to the new security framework, as well as suggestions for converting such users to a more manageable security implementation.

This section discusses the following topics regarding migration of privileges from an OFSA 3.5 or 4.0 database:

- Migration of Database Privilevges
- Migration of Application and Menu Privileges
- Guidelines for Managing Security Privileges of Migrated Users
- Removal of Security Filter

Migration of Database Privilevges

Database Privileges in OFSA 3.5 and 4.0

In OFSA 3.5 and 4.0, users received database privileges from the OFSA_USER role. All users were assigned to this role. The OFSA_USER role was a dynamic role, which was assigned SELECT, INSERT, UPDATE and DELETE privileges on all objects in the OFSA schema. The use of password encryption of user database passwords ensured the privileges for the OFSA_USER role were only available to users when logged into an OFS application. Because users did not know their *real* Oracle password, they were prevented from logging into Oracle Discoverer and other query tools with the user account granted the OFSA_USER role. Administrators created a second, External user account (no password encryption) for users requiring reporting and ad-hoc query privileges in Oracle Discoverer or other query tools.

In addition, the SQL Talk menu option within the OFS applications enables users to run query and update statements against objects in the database. Users with the

Select only restriction were allowed to run only SELECT queries, while users without this restriction were allowed to execute insert, update, and delete statements.

FDM 4.5 Database Privileges for Migrated Users

The following table outlines the privileges that users receive when migrated from an OFSA version 3.5 or 4.0 database.

Note: All database privileges for migrated users are assigned directly to their user account. Although this is contrary to the recommended approach of assigning privileges to User Groups and then assigning users to the User Groups to receive the privileges, it is unavoidable. Database privileges were not assigned to Groups in OFSA 3.5/4.0, so all migrations require direct assignment of these roles to the users.

User Category	4.5 Database Privileges assigned during upgrade process
All Users	Internal application role for each application to which user had access (example OFDM_R_PA for a user with access to Performance Analyzer. OFDM_R_CLIENT_PROC role for privileges on Client Data objects.
Users with SQL Talk Select Only	OFDM_R_REPORT_MART role OFDM_R_BUSINESS_PROCESS role
Users with SQL Talk Update privileges	OFDM_R_REPORT_MART role OFDM_R_BUSINESS_PROCESS role OFDM_R_CLIENT_EXT role

Migration of Application and Menu Privileges

Application and Menu Privileges in OFSA 3.5 and 4.0

In OFSA 3.5 and 4.0, users had no rights to login to an application unless specifically granted by the administrator. However, for menu privileges, users by default possessed all privileges for any application to which they had rights to login. Administrators then removed privileges from users that were restricted.

Application login privileges in OFSA 3.5/4.0 were therefore a union of all privileges the user received, either directly on the user account or from any Groups to which the user belonged. Menu access was a union of any menu restrictions assigned to the user, either directly or from a User Group.

FDM 4.5 Application and Menu (Function) Privileges for Migrated Users

The 4.5 Database Upgrade Process assigns all application and menu (referred to as *Function* privileges in 4.5) privileges directly to the migrated user accounts. Because 4.0 menu security was assigned as *restrictions* rather *grants of privileges*, it is impossible to accurately convert 3.5/4.0 Groups into 4.5 Security Profiles and maintain the same privilege level.

To complicate matters, the functionality embodied in Groups from OSFA 3.5 and 4.0 is now incorporated into 3 separate entities. The following table illustrates how a Group from OFSA 3.5 or 4.0 is migrated into the 4.5 structures:

Example: 3.5/4.0 Group name is RM_USERS

Entity Type	New 4.5 Entity Name	Notes
User Group	G_RM_USERS	No specific application or menu privileges. Assigned ID Folder access to the ID Folder created for the Group. Users from 3.5/4.0 remain members.
Security Profile	S_RM_USERS	Same privileges as the 3.5/4.0 Group. However, not attached to any users.
ID Folder	RM_USERS	All IDs from the 3.5/4.0 Group assigned to the new 4.5 ID Folder of the same name. ID Folder access assigned to the User Group created from the same 3.5/4.0 Group.

Because the Group concept in OFSA 3.5/4.0 embodied the functionality for 4.5 User Groups, Security Profiles and ID Folders, the Metadata Migration process creates separate entities from a single Group.

As noted in the table, the menu and application privileges previously assigned to the 3.5/4.0 Group are now assigned to a 4.5 Security Profile. However, the new Security Profile is not assigned to any users. Rather, users receive all of their privileges directly. The migration is performed in this manner because of the change in security paradigms from *union of restrictions* to *union of privileges*. The nature of this migration means that if users were assigned to the newly migrated Security Profile, they would possess *more* privileges than they originally had in 3.5/4.0.

For example, if a user was previously restricted at the user level from opening Performance Analyzer Allocations but was a member of a 3.5/4.0 Group that did not have this restriction, the restriction would still be in place for the user in OFSA 3.5 or 4.0. This is because OFSA 3.5/4.0 implements security as a *union of restrictions*. However, FDM 4.5 implements security as a *union of privileges*. Therefore, the 4.5 Database Upgrade Process does not assign users to any of the newly migrated Security Profiles. To do so would result in the users possessing more privileges than they originally had in OFSA 3.5/4.0. Rather, the database upgrade process assigns all privileges directly to the users, regardless of the original source of the privileges. The new Security Profiles are created only for the contingency that administrators might desire to use them in the future.

To summarize:

Users

- All 4.5 privileges are assigned directly to the user.
- Application login privileges are a union of all application login rights received in 3.5/4.0, whether directly or from a 3.5/4.0 Group.
- Function privileges are a union of all menu privileges not restricted from the user in 3.5/4.0, whether directly or from a 3.5/4.0 Group.
- Assigned as members of the User Groups that were migrated from 3.5/4.0 Groups.
- Receive ID Folder access from membership in the User Groups migrated from 3.5/4.0.

Groups

- Migrated into 3 new entities - a User Group, a Security Profile, and an ID Folder.
- Application and menu privileges are migrated to the new Security Profile. However, the new Security Profile is not assigned to any users.

- OFSA IDs are migrated from the Group to the new ID Folder. The new ID Folder is assigned to the corresponding User Group.
- The User Group has no menu or application privileges. However, the User Group is assigned access to the ID Folder. All users that were members of the 3.5/4.0 Group are assigned as members of the new User Group.

Guidelines for Managing Security Privileges of Migrated Users

Oracle recommends that you reorganize security for any users migrated from OFSA version 3.5 or 4.0. Application and Function privileges for these users in 4.5 are direct assignments, thereby creating additional management overhead for maintaining their security access. A more effective security management strategy for these users would be to use the new Security Profile entity in 4.5, in conjunction with User Groups, to simplify security assignments.

Use the seeded User Groups, Security Profiles and Roles if possible. For example, assign users requiring access to Performance Analyzer to the OFDM_G_PA User Group. This provides them with the seeded application role as well as the seeded Security Profile for Performance Analyzer. However, because the seeded User Groups and Security Profiles provide full function privileges to each application, you may need to create customized entities to provide the appropriate privileges.

To do this, identify the different categories of users that access the FDM database. For each different category of user, create a corresponding Security Profile to manage application and function privileges. Assign users requiring the same privileges to a single User Group, and assign the Security Profile for these users to that same User Group. Also assign the seeded application role (such as OFDM_R_PA) and any appropriate Client Data roles to the User Group. In this manner, the privilege assignments are greatly simplified. When you need to modify a function privilege assignment for the entire category of users, all you need to do is make the modification to the single Security Profile. If you need to modify the database privileges for the group of users, you need only modify one of the Client Data Roles, or perhaps created a new customized role.

To summarize, here are some guidelines for security maintenance in 4.5:

1. Assign users to seeded User Groups whenever possible.
2. Create customized Security Profiles for each category of users requiring separate privileges.
3. Create customized Roles for access to Client Data objects.

4. Assign Security Profiles and Roles to the customized User Group for each category of users. Assign the seeded application roles (such as OFDM_R_PA) to the appropriate User Groups.
5. Assign ID Folders to the User Groups to provide users with access to OFSA IDs.
6. Remove application and function privileges assigned to the users directly by the 4.5 Database Upgrade Process.

Removal of Security Filter

FDM Administration version 4.5 does not include a Security Data Filter feature that was present in the OFSA 3.5/4.0 System Administration application. Rather, FDM 4.5 relies upon the native Fine Grain Access Control features within the Oracle DBMS_RLS package. Refer to the *Oracle8i Application Developers Guide* for more information regarding Fine Grain Access Control features and the DBMS_RLS package.

15

FDM Multi-Language Support

The Oracle Financial Data Manager (FDM) database environment supports the use of multiple languages (otherwise known as Multi-Language Support or MLS). Within FDM, Multi-Language Support means that the translatable information, such as descriptions or display names, is stored in a separate table from the other attributes of that object. Users then access the data from a database view (termed a Language Compatible View) joining the translatable information with the base attributes of the object.

> **Note:** Although Release 4.5 of the FDM database provides the foundation for supporting multiple languages, the individual applications in the Oracle Financial Services Applications (OFSA) group of applications are not enabled for Multi-Language Support for Release 4.5. Also, no translations of the FDM-required seeded data into non-English languages are available at the publication time of this guide. The Multi-Language Support capabilities of the 4.5 FDM database environment are available for custom purposes only.

Multi-Language Support within FDM enables users of multiple languages to retrieve information in their own language from the same database. For example, a German user logged into the Performance Analyzer application views display names for FDM tables and columns in German, while an English user logged into the very same database views display names in English. The FDM database stores display name translations in both languages simultaneously. The user only views the translatable values for his or her specific language (based upon the language installed upon the user's client PC).

This chapter includes the following topics to describe how multiple language support is implemented within the FDM database:

- Session Language
- MLS Database Structures
- Creating MLS Objects
- Seeded MLS Objects

Note: FDM 4.5 only supports multiple languages in the same database character set. The Unicode character set is not supported.

Note: Although all seeded FDM tables with translatable values are MLS-enabled, you are not required to MLS-enable any user-defined objects that you create for FDM. The flexibility of the FDM Metadata enables you to define any of your own objects for single language use only.

Session Language

The Session Language is the language that translatable values appear to the user. For example, an English user views translatable information in English, while a German user views translatable information in German.

The Session Language for an operation is determined based upon the origin of that operation:

Operation Origin	Session Language
Client PC based	NLS_LANG parameter in the registry.
Server based	NLS_LANG environment variable on the server
Browser based (in 3 tier architecture)	NLS_LANG parameter on the application server

Thus, the language for a user running a report from Discoverer on a client PC is determined based upon the NLS_LANG parameter in the registry on that PC. The language for the log file for an Allocation process run on the server is determined by the NLS_LANG environment variable on that server. The language for a query

run from a browser based application is determined by the NLS_LANG parameter on the application server to which the browser is accessing.

MLS Database Structures

FDM implements Multi-Language Support in Release 4.5 by separating translatable information (such as written descriptions) from attributes (such as column attributes). The translatable information is stored in an MLS table while the attribute information is stored in a Base table. The user or application then retrieves data from a Language Compatible view that joins the Base table information with the translatable information in the MLS table.

The OFSA_MLS Table

The OFSA_MLS table identifies all of the languages installed in the database. The structure of this table is as follows:

```
MLS_CD                                    NOT NULL VARCHAR2(3)
INSTALLED_FLG                             NOT NULL NUMBER(1)
LANGUAGE                                  NOT NULL VARCHAR2(30)
DESCRIPTION                                        VARCHAR2(255)
```

The MLS_CD field identifies the language for translatable values in the MLS tables in the database. The OFSA_MLS table is populated by the FDM Database Creation Process. An INSTALLED_FLG = 1 indicates that translatable values for that language were seeded for all FDM MLS tables by the Database Creation Process.

> **Note:** The INSTALLED_FLG is set by the FDM Database Creation or FDM Database Upgrade Processes. The INSTALLED_FLG must be set to 1 for a language to be available for use with FDM.

Base Tables

The Base Table stores all of the non-translatable attributes for an entity. For example, if you have an Account Officer Type Code where there is a flag identifying if the Account Officer Type is active, this would be considered a Base table attribute.

The FDM naming convention is that the Base table is the name of the entity. So, for the FDM Table Classifications, the base table is OFSA_TABLE_CLASSIFICATION. This convention is further refined in the case of seeded Code tables. The base table

for all seeded Codes is named _CD. So, for the table storing Accrual Basis Codes, the base table name is OFSA_ACCRUAL_BASIS_CD.

For seeded Code Description tables, there are no base table attributes other than the actual code value. Therefore, the base table for these entities contains only the single code column. For example, the base table for OFSA_ACCRUAL_BASIS contains only the single column, ACCRUAL_BASIS_CD.

MLS Tables

MLS Tables store all of the translatable attributes for an entity. In the Account Officer Code example, the display names and descriptions for Account Officer Types are stored in (a user-defined table) ACCOUNT_OFFICER_TYPE_MLS, which is separate from the Base table.

MLS tables are always fully populated based upon the languages installed in the database. In each MLS table, there is always one record per installed language for each base table record. In other words, every installed language always has a full compliment of translatable records for each base table record. FDM maintains this fully populated state by inserting and deleting rows as required using database triggers. For new inserts, the translatable columns in the MLS table are populated with defaults from the base table.

The FDM Database Creation and Database Upgrade processes fully populate all MLS tables. The triggers on the Language Compatible Views maintain this fully populated state for any new inserts or deletes. For inserts, the triggers provide default values for the MLS tables. These need to be translated into the appropriate languages afterwards.

The FDM naming convention is that all tables storing translatable values are named with the entity name followed by an _MLS suffix. So, for the Accrual Basis Code example, the MLS table is named OFSA_ACCRUAL_BASIS_MLS.

For details on how to create the appropriate triggers for maintaining a fully populated state for user-defined MLS objects, refer to Creating MLS Objects.

Language Compatible Views

Each Base table and MLS table have a corresponding Language Compatible view for all SQL operations for that entity. The Language Compatible View joins the Base table with the MLS table to provide a single object from which the user or application queries. In addition, the Language Compatible View is constructed using the Oracle USERENV function to filter on the user's specified language. This

enables a German user to retrieve translatable results in German, while an English user retrieves translatable results in English.

Note: Because Language Compatible Views support SELECT, INSERT, UPDATE and DELETE operations, and because they are language aware, use these objects for any user or application SQL operations. Do not perform SQL operations against Base tables or MLS tables directly.

Note: The Language Compatible Views described in this chapter provide updates to the attribute columns (such as DISPLAY_NAME or DESCRIPTION), but not to the Code column itself (such as ACCRUAL_BASIS_CD). These views are defined in such a manner that if you need to update values for the Code column, you must delete the appropriate records and then re-insert with the corrected values.

The USERENV function reads the NLS_LANG registry entry (or environment variable if you are on UNIX) to identify the language for the session. This registry entry is created during the installation of the application. Queries originating from a client PC reference the PC registry entry, while queries originating from a server (including an application server) reference either the registry entry or a UNIX environment variable, depending upon the platform. For more information about the USERENV function, refer to the appropriate Oracle RDMBS reference material.

For an example of how to create a Language Compatible View, refer to Creating MLS Objects.

Creating MLS Objects

FDM enables you to create and register multi-support enabled, user-defined database objects. The seeded FDM Metadata and Code Description entities are already enabled for Multi-Language Support. However, if you are in a multi-lingual environment, you may want to MLS-enable user-defined database objects that you create.

FDM enables you to MLS-enable any user-defined table structures.

> **Note:** If you are operating in a multi-language environment, you should MLS-enable all user-defined Code Description tables. With few exceptions, Code Description tables are the only user-defined tables storing purely translatable information.

This section outlines the required steps to create MLS-enabled data structures. Because the most common example of creating new MLS-enabled objects is for maintaining user-defined Code Descriptions, this section uses a Code Description object for example purposes.

Required steps for MLS-enabled database structures:

1. Create the Base Table
2. Create the MLS table
3. Create the Language Compatible View
4. Create database triggers
5. Register the objects in FDM Administration

Each of these steps is discussed in detail, as follows:

Create the Base Table

For Code Description data structures, the Base table usually consists of only the code column. In other situations you can have other base table attributes.

An example Base table is the OFSA_ACCRUAL_BASIS_CD table. This table has a single column ACCRUAL_BASIS_CD.

Create the MLS Table

The MLS table contains all of the translatable columns and an MLS code value for identifying the language. For Code Descriptions, the translatable columns usually consist of a translatable name for the Code as well as a long description.

An example MLS table is the OFSA_ACCRUAL_BASIS_MLS table. This table has the following structure:

```
MLS_CD                                  NOT NULL VARCHAR2(3)
ACCRUAL_BASIS_CD                        NOT NULL NUMBER(5)
ACCRUAL_BASIS                           NOT NULL VARCHAR2(40)
```

```
DESCRIPTION                                    VARCHAR2(255)
```

Create the Language Compatible View

The Language Compatible View is the object that joins the base table attributes to the translatable values in the MLS table. The Language Compatible View filters on the user's specified language so that any queries return translatable values in only that language. When all of the appropriate database triggers are in place, this view can be used for Select, Insert, Update and Delete operations.

An example of a Language Compatible View is the OFSA_ACCRUAL_BASIS_DSC view. This view was created using the following SQL:

```
PROMPT Creating View 'OFSA_ACCRUAL_BASIS_DSC'
CREATE OR REPLACE FORCE VIEW OFSA_ACCRUAL_BASIS_DSC
 (ACCRUAL_BASIS_CD
 ,ACCRUAL_BASIS
 ,DESCRIPTION)
 AS SELECT
        ACCRULBSML.ACCRUAL_BASIS_CD ACCRUAL_BASIS_CD,
        ACCRULBSML.ACCRUAL_BASIS ACCRUAL_BASIS,
        ACCRULBSML.DESCRIPTION DESCRIPTION
FROM OFSA_ACCRUAL_BASIS_MLS ACCRULBSML
  WHERE MLS_CD = USERENV('LANG')
```

When queried, this view returns only records in the designated language for that session.

Create the Database Triggers

Database triggers support insert, update and delete operations against the Language Compatible View, and also maintain the fully populated state of the MLS tables.

Examples of the triggers that are required for the Accrual Basis Codes data structures are as follows.

Base Table Trigger

This trigger maintains the fully populated state. Whenever a new Code value is inserted into the Base table, this trigger fires to insert a record into the MLS table for each installed language.

```
-- AFTER INSERT trigger on OFSA_ACCRUAL_BASIS_CD.
-- For each row INSERTed into OFSA_ACCRUAL_BASIS_CD,
-- INSERT one row into OFSA_ACCRUAL_BASIS_MLS for each
```

```
-- installed language.
DECLARE
  CURSOR c1 IS
    SELECT mls_cd
    FROM ofsa_mls WHERE installed_flg=1;
BEGIN
   FOR installed_languages IN c1 LOOP
      INSERT INTO ofsa_accrual_basis_mls
        (mls_cd, accrual_basis_cd,
         accrual_basis,description)
      VALUES
        (installed_languages.mls_cd, :new.accrual_basis_cd,
         :new.accrual_basis_cd, :new.accrual_basis_cd);
   END LOOP;
END;
```

Language Compatible View Trigger

This trigger supports insert, update and delete operations against the Language Compatible View. This simplifies the implementation so that all SQL operations can be executed against the database view.

```
-- ****************************************************************
-- ** INSTEAD-OF INSERT trigger on OFSA_ACCRUAL_BASIS_DSC (view). **
-- ****************************************************************
-- For each row, instead of inserting into OFSA_ACCRUAL_BASIS_DSC,
-- insert INTO OFSA_ACCRUAL_BASIS_CD. This will fire
-- the INSERT trigger on OFSA_ACCRUAL_BASIS_CD which INSERTs
-- one row into OFSA_ACCRUAL_BASIS_MLS for each installed
-- language.  THEN this trigger updates OFSA_ACCRUAL_BASIS_MLS
-- for the current language.
IF INSERTING THEN
   INSERT INTO ofsa_accrual_basis_cd
     (accrual_basis_cd)
   VALUES
     (:new.accrual_basis_cd );

   UPDATE ofsa_accrual_basis_mls
   SET description   = :new.description,
       accrual_basis = :new.accrual_basis
   WHERE mls_cd         = USERENV('LANG')
   AND accrual_basis_cd = :new.accrual_basis_cd;

-- ****************************************************************
-- ** INSTEAD-OF DELETE trigger on OFSA_ACCRUAL_BASIS_DSC (view). **
```

```
-- ****************************************************************
-- For each row, instead of deleting from OFSA_ACCRUAL_BASIS_DSC,
-- delete FROM OFSA_ACCRUAL_BASIS_CD AND this
-- table will delete cascade and remove all the entries in
-- OFSA_ACCRUAL_BASIS_MLS
ELSIF DELETING THEN
   DELETE FROM ofsa_accrual_basis_cd
   WHERE accrual_basis_cd = :old.accrual_basis_cd;

END IF;
```

Register Objects in FDM Administration

Once all of the required database objects have been created in the database, you need to register them in FDM Administration. Objects registered in the FDM Metadata are then available for user operations.

Table Classification Assignments

Assign an appropriate Table Classification to the newly registered objects: For MLS-enabled Code Description objects, create the following Table Classification assignments:

MLS Object	Table Classification Assignment
Base Table	295 - Codes User-Defined
MLS Table	296 - MLS Descriptions User-Defined
Language Compatible View	298 - Code Descriptions User-Defined

For more information regarding these Table Classifications, refer to Chapter 16, "FDM Object Management". For information on how to assign a Table Classification to an object, refer to the *Oracle Financial Data Manager Administration Guide.*

Description Table Mapping

You may need to map any Code columns that exist on instrument and other client data objects to the appropriate Language Compatible View. To do so, you use the Description Table Mapping Wizard within FDM Administration. This enables the translated names and descriptions to display in place of the code value. For more

information regarding how to perform this mapping, refer to the *Oracle Financial Data Manager Administration Guide*.

Seeded MLS Objects

All FDM tables storing translatable seeded data are MLS-enabled in Release 4.5. There are two categories of MLS-enabled tables in FDM:

- FDM Metadata tables
- FDM seeded Code Description tables

Display names that are user specific names are not translated. Examples include the names and descriptions of IDs (stored in OFSA_CATALOG_OF_IDS) as well as individual Leaf values (except for Financial Elements).

FDM Metadata Tables

FDM supports display names and descriptions in multiple languages for all FDM registered tables. User's logged into the OFS applications view any FDM registered objects in their specified language.

Display names and descriptions for FDM Table Classifications and Financial Element Leaf values are also MLS-enabled.

Code Description Tables

A Code Description table is a table that stores a written description for a numeric or alphanumeric code value. For example, the written description for the code value USD is US Dollars.

All seeded Code Description tables are MLS-enabled. However, the FDM Database Upgrade Process does not MLS-enable any user-defined Code Description tables. Seeded MLS Objects

The following table lists the objects enabled for Multi-Language Support seeded by the FDM Database Creation and Database Upgrade processes. With few exceptions, all Base tables and MLS tables have a corresponding Language Compatible View:

Base Table	MLS Table	Language Compatible View
ASSIGNMENT_METHOD_CD	ASSIGNMENT_METHOD_MLS	ASSIGNMENT_METHOD_DSC
CAMPAIGN_CALC_SOURCE_CD	CAMPAIGN_CALC_SOURCE_MLS	CAMPAIGN_CALC_SOURCE_DSC

Base Table	MLS Table	Language Compatible View
CAMPAIGN_MEASURE_CD	CAMPAIGN_MEASURE_MLS	CAMPAIGN_MEASURE_DSC
COLLATERAL_DISCHARGE_TYPE_CD	COLLATERAL_DISCHARGE_TYPE_MLS	COLLATERAL_DISCHARGE_TYPE_DSC
COLLATERAL_RELATIONSHIP_CD	COLLATERAL_RELATIONSHIP_MLS	COLLATERAL_RELATIONSHIP_DSC
COLLATERAL_STATUS_CD	COLLATERAL_STATUS_MLS	COLLATERAL_STATUS_DSC
COLLATERAL_SUB_TYPE_CD	COLLATERAL_SUB_TYPE_MLS	COLLATERAL_SUB_TYPE_DSC
CONTACT_METHOD_CD	CONTACT_METHOD_MLS	CONTACT_METHOD_DSC
INCENTIVE_TYPE_CD	INCENTIVE_TYPE_MLS	INCENTIVE_TYPE_DSC
OFSA_ACCRUAL_BASIS_CD	OFSA_ACCRUAL_BASIS_MLS	OFSA_ACCRUAL_BASIS_DSC
OFSA_ACCUMULATION_TYPE_CD	OFSA_ACCUMULATION_TYPE_MLS	OFSA_ACCUMULATION_TYPE_DSC
OFSA_ADJUSTABLE_TYPE_CD	OFSA_ADJUSTABLE_TYPE_MLS	OFSA_ADJUSTABLE_TYPE_DSC
OFSA_AMORTIZATION_TYPE_CD	OFSA_AMORTIZATION_TYPE_MLS	OFSA_AMORTIZATION_TYPE_DSC
OFSA_AMOUNT_TYPE_CD	OFSA_AMOUNT_TYPE_MLS	OFSA_AMOUNT_TYPE_DSC
OFSA_BATCH_EVENTS_STATUS_CD	OFSA_BATCH_EVENTS_STATUS_MLS	OFSA_BATCH_EVENTS_STATUS_DSC
OFSA_BATCH_EVENTS_TYPE_CD	OFSA_BATCH_EVENTS_TYPE_MLS	OFSA_BATCH_EVENTS_TYPE_DSC
OFSA_CALC_SOURCE_CD	OFSA_CALC_SOURCE_MLS	OFSA_CALC_SOURCE_DSC
OFSA_CMO_TRANCHE_CD	OFSA_CMO_TRANCHE_MLS	OFSA_CMO_TRANCHE_DSC
OFSA_COLLATERAL_CD	OFSA_COLLATERAL_MLS	OFSA_COLLATERAL_DSC
OFSA_COLUMN_PROPERTY_CD	OFSA_COLUMN_PROPERTY_MLS	OFSA_COLUMN_PROPERTY_DSC
OFSA_COLUMN_REQUIREMENTS	OFSA_COLUMN_REQUIREMENTS_MLS	
OFSA_COMMITMENT_TYPE_CD	OFSA_COMMITMENT_TYPE_MLS	OFSA_COMMITMENT_TYPE_DSC
OFSA_COMPONENT_TYPE_CD	OFSA_COMPONENT_TYPE_MLS	OFSA_COMPONENT_TYPE_DSC
OFSA_COMPOUND_BASIS_CD	OFSA_COMPOUND_BASIS_MLS	OFSA_COMPOUND_BASIS_DSC
OFSA_CONFORMANCE_CD	OFSA_CONFORMANCE_MLS	OFSA_CONFORMANCE_DSC
OFSA_CONSOLIDATION_CD	OFSA_CONSOLIDATION_MLS	OFSA_CONSOLIDATION_DSC
OFSA_CORRECTION_PROC_MSG_CD	OFSA_CORRECTION_PROC_MSG_MLS	OFSA_CORRECTION_PROC_MSG_DSC
OFSA_CREDIT_RATING_CD	OFSA_CREDIT_RATING_MLS	OFSA_CREDIT_RATING_DSC
OFSA_CREDIT_STATUS_CD	OFSA_CREDIT_STATUS_MLS	OFSA_CREDIT_STATUS_DSC
OFSA_CURRENCIES	OFSA_CURRENCY_MLS	OFSA_CURRENCIES_V
OFSA_CURRENCY_STATUS_CD	OFSA_CURRENCY_STATUS_MLS	OFSA_CURRENCY_STATUS_DSC

Base Table	MLS Table	Language Compatible View
OFSA_DETAIL_ELEM_B	OFSA_DETAIL_ELEM_MLS	OFSA_DETAIL_ELEM
OFSA_DETAIL_RECORD_CD	OFSA_DETAIL_RECORD_MLS	OFSA_DETAIL_RECORD_DSC
OFSA_DIRECT_IND_CD	OFSA_DIRECT_IND_MLS	OFSA_DIRECT_IND_DSC
OFSA_DISCOUNT_RATE_METHOD_CD	OFSA_DISCOUNT_RATE_METHOD_MLS	OFSA_DISCOUNT_RATE_METHOD_DSC
OFSA_ESTIMATION_SMOOTHING_CD	OFSA_ESTIMATION_SMOOTHING_MLS	OFSA_ESTIMATION_SMOOTHING_DSC
OFSA_EXCHANGE_RATE_STATUS_CD	OFSA_EXCHANGE_RATE_STATUS_MLS	OFSA_EXCHANGE_RATE_STATUS_DSC
OFSA_EXCHNG_RATE_CONV_TYPE_CD	OFSA_EXCHNG_RATE_CONV_TYPE_MLS	OFSA_EXCHNG_RATE_CONV_TYPE_DSC
OFSA_EXIST_BORROWER_CD	OFSA_EXIST_BORROWER_MLS	OFSA_EXIST_BORROWER_DSC
OFSA_FBAL_BOOKING_CD	OFSA_FBAL_BOOKING_MLS	OFSA_FBAL_BOOKING_DSC
OFSA_FBAL_DIMENSION_CD	OFSA_FBAL_DIMENSION_MLS	OFSA_FBAL_DIMENSION_DSC
OFSA_FBAL_METHOD_CD	OFSA_FBAL_METHOD_MLS	OFSA_FBAL_METHOD_DSC
OFSA_FBAL_RATE_VOLUME_CD	OFSA_FBAL_RATE_VOLUME_MLS	OFSA_FBAL_RATE_VOLUME_DSC
OFSA_FBAL_RUNOFF_CD	OFSA_FBAL_RUNOFF_MLS	OFSA_FBAL_RUNOFF_DSC
OFSA_FCAST_IRC_METHOD_CD	OFSA_FCAST_IRC_METHOD_MLS	OFSA_FCAST_IRC_METHOD_DSC
OFSA_FCAST_XRATE_METHOD_CD	OFSA_FCAST_XRATE_METHOD_MLS	OFSA_FCAST_XRATE_METHOD_DSC
OFSA_FINANCIAL_SCENARIO_CD	OFSA_FINANCIAL_SCENARIO_MLS	OFSA_FINANCIAL_SCENARIO_DSC
OFSA_FORWARD_TYPE_CD	OFSA_FORWARD_TYPE_MLS	OFSA_FORWARD_TYPE_DSC
OFSA_FREQUENCY_UNIT_CD	OFSA_FREQUENCY_UNIT_MLS	OFSA_FREQUENCY_UNIT_DSC
OFSA_GEOGRAPHIC_LOC_CD	OFSA_GEOGRAPHIC_LOC_MLS	OFSA_GEOGRAPHIC_LOC_DSC
OFSA_HELD_FOR_SALE_CD	OFSA_HELD_FOR_SALE_MLS	OFSA_HELD_FOR_SALE_DSC
OFSA_ID_TYPE_CD	OFSA_ID_TYPE_MLS	OFSA_ID_TYPE_DSC
OFSA_INSTRUMENT_TYPE_CD	OFSA_INSTRUMENT_TYPE_MLS	OFSA_INSTRUMENT_TYPE_DSC
OFSA_INSURANCE_TYPE_CD	OFSA_INSURANCE_TYPE_MLS	OFSA_INSURANCE_TYPE_DSC
OFSA_INTEREST_TIMING_TYPE_CD	OFSA_INTEREST_TIMING_TYPE_MLS	OFSA_INTEREST_TIMING_TYPE_DSC
OFSA_INT_COMPONENT_TYPE_CD	OFSA_INT_COMPONENT_TYPE_MLS	OFSA_INT_COMPONENT_TYPE_DSC
OFSA_IRC_FORMAT_CD	OFSA_IRC_FORMAT_MLS	OFSA_IRC_FORMAT_DSC
OFSA_ISSUER_CD	OFSA_ISSUER_MLS	OFSA_ISSUER_DSC
OFSA_JOB_STATUS_CD	OFSA_JOB_STATUS_MLS	OFSA_JOB_STATUS_DSC

Base Table	MLS Table	Language Compatible View
OFSA_LIEN_POSITION_CD	OFSA_LIEN_POSITION_MLS	OFSA_LIEN_POSITION_DSC
OFSA_LIQUIDITY_CLASS_CD	OFSA_LIQUIDITY_CLASS_MLS	OFSA_LIQUIDITY_CLASS_DSC
OFSA_LOAN_PROPERTY_TYPE_CD	OFSA_LOAN_PROPERTY_TYPE_ MLS	OFSA_LOAN_PROPERTY_TYPE_ DSC
OFSA_MARKET_SEGMENT_CD	OFSA_MARKET_SEGMENT_MLS	OFSA_MARKET_SEGMENT_DSC
OFSA_MESSAGES_B	OFSA_MESSAGES_MLS	OFSA_MESSAGES
OFSA_MODIFY_ACTION_CD	OFSA_MODIFY_ACTION_MLS	OFSA_MODIFY_ACTION_DSC
OFSA_MORTGAGE_AGENCY_CD	OFSA_MORTGAGE_AGENCY_MLS	OFSA_MORTGAGE_AGENCY_DSC
OFSA_MSG_SEVERITY_CD	OFSA_MSG_SEVERITY_MLS	OFSA_MSG_SEVERITY_DSC
OFSA_MULTIPLIER_CD	OFSA_MULTIPLIER_MLS	OFSA_MULTIPLIER_DSC
OFSA_NET_MARGIN_CD	OFSA_NET_MARGIN_MLS	OFSA_NET_MARGIN_DSC
OFSA_OCCUPANCY_CD	OFSA_OCCUPANCY_MLS	OFSA_OCCUPANCY_DSC
OFSA_OPTION_EXERCISE_CD	OFSA_OPTION_EXERCISE_MLS	OFSA_OPTION_EXERCISE_DSC
OFSA_OPTION_TYPE_CD	OFSA_OPTION_TYPE_MLS	OFSA_OPTION_TYPE_DSC
OFSA_OVERDRAFT_PROTECTION_ CD	OFSA_OVERDRAFT_PROTECTION_ MLS	OFSA_OVERDRAFT_PROTECTION_ DSC
OFSA_OWNERSHIP_CD	OFSA_OWNERSHIP_MLS	OFSA_OWNERSHIP_DSC
OFSA_PATTERN_TYPE_CD	OFSA_PATTERN_TYPE_MLS	OFSA_PATTERN_TYPE_DSC
OFSA_PAYMENT_TYPE_CD	OFSA_PAYMENT_TYPE_MLS	OFSA_PAYMENT_TYPE_DSC
OFSA_PLEDGED_STATUS_CD	OFSA_PLEDGED_STATUS_MLS	OFSA_PLEDGED_STATUS_DSC
OFSA_PMT_PATTERN_TYPE_CD	OFSA_PMT_PATTERN_TYPE_MLS	OFSA_PMT_PATTERN_TYPE_DSC
OFSA_PP_CALC_METHOD_CD	OFSA_PP_CALC_METHOD_MLS	OFSA_PP_CALC_METHOD_DSC
OFSA_PP_DIM_TYPE_CD	OFSA_PP_DIM_TYPE_MLS	OFSA_PP_DIM_TYPE_DSC
OFSA_PP_QUOTE_CD	OFSA_PP_QUOTE_MLS	OFSA_PP_QUOTE_DSC
OFSA_PP_RATE_TERM_CD	OFSA_PP_RATE_TERM_MLS	OFSA_PP_RATE_TERM_DSC
OFSA_PRIVATE_MTG_INSURER_ CD	OFSA_PRIVATE_MTG_INSURER_ MLS	OFSA_PRIVATE_MTG_INSURER_ DSC
OFSA_PROCESS_FILTER_TYPE_CD	OFSA_PROCESS_FILTER_TYPE_ MLS	OFSA_PROCESS_FILTER_TYPE_ DSC
OFSA_PROCESS_PARTITION_CD	OFSA_PROCESS_PARTITION_MLS	OFSA_PROCESS_PARTITION_DSC
OFSA_PRODUCT_TYPE_CD	OFSA_PRODUCT_TYPE_MLS	OFSA_PRODUCT_TYPE_DSC
OFSA_PURPOSE_CD	OFSA_PURPOSE_MLS	OFSA_PURPOSE_DSC
OFSA_PUT_CALL_CD	OFSA_PUT_CALL_MLS	OFSA_PUT_CALL_DSC
OFSA_RATE_CAP_TYPE_CD	OFSA_RATE_CAP_TYPE_MLS	OFSA_RATE_CAP_TYPE_DSC

Base Table	MLS Table	Language Compatible View
OFSA_RATE_CHG_ROUNDING_CD	OFSA_RATE_CHG_ROUNDING_MLS	OFSA_RATE_CHG_ROUNDING_DSC
OFSA_RATE_DATA_SOURCE_CD	OFSA_RATE_DATA_SOURCE_MLS	OFSA_RATE_DATA_SOURCE_DSC
OFSA_RATE_FLOOR_TYPE_CD	OFSA_RATE_FLOOR_TYPE_MLS	OFSA_RATE_FLOOR_TYPE_DSC
OFSA_REG_D_STATUS_CD	OFSA_REG_D_STATUS_MLS	OFSA_REG_D_STATUS_DSC
OFSA_REPRICE_METHOD_CD	OFSA_REPRICE_METHOD_MLS	OFSA_REPRICE_METHOD_DSC
OFSA_RESULT_TYPE_CD	OFSA_RESULT_TYPE_MLS	OFSA_RESULT_TYPE_DSC
OFSA_ROLL_FACILITY_CD	OFSA_ROLL_FACILITY_MLS	OFSA_ROLL_FACILITY_DSC
OFSA_SERVICING_AGENT_CD	OFSA_SERVICING_AGENT_MLS	OFSA_SERVICING_AGENT_DSC
OFSA_SETTLEMENT_TYPE_CD	OFSA_SETTLEMENT_TYPE_MLS	OFSA_SETTLEMENT_TYPE_DSC
OFSA_SIC_CD	OFSA_SIC_MLS	OFSA_SIC_DSC
OFSA_SMOOTHING_METHOD_CD	OFSA_SMOOTHING_METHOD_MLS	OFSA_SMOOTHING_METHOD_DSC
OFSA_SOLICIT_SOURCE_CD	OFSA_SOLICIT_SOURCE_MLS	OFSA_SOLICIT_SOURCE_DSC
OFSA_STOCH_RANDOM_SEQ_TYPE_CD	OFSA_STOCH_RANDOM_SEQ_TYPE_MLS	OFSA_STOCH_RANDOM_SEQ_TYPE_DSC
OFSA_STRIKE_TYPE_CD	OFSA_STRIKE_TYPE_MLS	OFSA_STRIKE_TYPE_DSC
OFSA_STRINGS_B	OFSA_STRINGS_MLS	OFSA_STRINGS
OFSA_TABLES	OFSA_TABLES_MLS	OFSA_TABLES_V
OFSA_TABLE_CLASSIFICATION	OFSA_TABLE_CLASSIFICATION_MLS	OFSA_TABLE_CLASSIFICATION_DSC
OFSA_TAB_COLUMNS	OFSA_TAB_COLUMNS_MLS	OFSA_TAB_COLUMNS_V
OFSA_TERM_TYPE_CD	OFSA_TERM_TYPE_MLS	OFSA_TERM_TYPE_DSC
OFSA_TM_PROC_TYPE_CD	OFSA_TM_PROC_TYPE_MLS	OFSA_TM_PROC_TYPE_DSC
OFSA_TP_ASSIGN_DATE_CD	OFSA_TP_ASSIGN_DATE_MLS	OFSA_TP_ASSIGN_DATE_DSC
OFSA_TP_CALC_METHOD_CD	OFSA_TP_CALC_METHOD_MLS	OFSA_TP_CALC_METHOD_DSC
OFSA_TP_CALC_MODE_CD	OFSA_TP_CALC_MODE_MLS	OFSA_TP_CALC_MODE_DSC
OFSA_TP_LEAF_DATA_SOURCE_CD	OFSA_TP_LEAF_DATA_SOURCE_MLS	OFSA_TP_LEAF_DATA_SOURCE_DSC
OFSA_TP_OPT_COST_METHOD_CD	OFSA_TP_OPT_COST_METHOD_MLS	OFSA_TP_OPT_COST_METHOD_DSC
OFSA_TP_TARGET_BAL_CD	OFSA_TP_TARGET_BAL_MLS	OFSA_TP_TARGET_BAL_DSC
OFSA_TRANSFORM_PROC_SCOPE_CD	OFSA_TRANSFORM_PROC_SCOPE_MLS	OFSA_TRANSFORM_PROC_SCOPE_DSC
OFSA_TRANSFORM_SRC_TYPE_CD	OFSA_TRANSFORM_SRC_TYPE_MLS	OFSA_TRANSFORM_SRC_TYPE_DSC

Base Table	MLS Table	Language Compatible View
OFSA_TS_MODEL_CD	OFSA_TS_MODEL_MLS	OFSA_TS_MODEL_DSC
OFSA_USAGE_CD	OFSA_USAGE_MLS	OFSA_USAGE_DSC
OFSA_VIRTUAL_TABLES	OFSA_VIRTUAL_TABLES_MLS	(there is no DSC view for these tables)
QUERY_ROLE_CD	QUERY_ROLE_MLS	QUERY_ROLE_DSC
QUERY_SOURCE_CD	QUERY_SOURCE_MLS	QUERY_SOURCE_DSC
RESPONSIBLE_PARTY_CD	RESPONSIBLE_PARTY_MLS	RESPONSIBLE_PARTY_DSC
TRACKING_METHOD_CD	TRACKING_METHOD_MLS	TRACKING_METHOD_DSC
TRACKING_STATUS_CD	TRACKING_STATUS_MLS	TRACKING_STATUS_DSC

16

FDM Object Management

This chapter provides information on how to manage objects within the Oracle Financial Data Manager (FDM) environment. FDM Object management encompasses the creation and customization of database tables and views for use with the FDM database, as well as use with other applications in the Oracle Financial Services Applications (OFSA) group of applications. The FDM database is the foundation for the OFS applications. To use database objects for OFS application operations, those objects must be identified properly within the FDM metadata.

The following, specific topics are covered in this chapter:

- FDM Database Environment
- Object Registration
- Client Data Objects
- Risk Manager Results Tables
- Transformation Output Tables
- Temporary Objects
- Message and Audit Objects
- Packages, Procedures, and Java Classes
- Views and Triggers
- Seeded Data Tables and Ranges

FDM Database Environment

The FDM database environment encompasses all of the database tables and views registered within the FDM Metadata, as well as supporting objects such as triggers,

constraints, indexes and public synonyms. This environment also includes objects providing security, such as users and roles.

FDM supports customization of this environment. However, objects with the OFSA_ prefix are categorized as FDM Reserved. FDM Reserved objects support the internal operations of the OFS applications.With few exceptions, these objects cannot be customized or modified in any manner. The exceptions to this rule include modifying FDM Reserved Objects for the purpose of registering a new Leaf Column.

In general, objects without the OFSA_ prefix are categorized as Client Data Objects. Client Data Objects are completely customizable. You can also create your own objects for use with FDM. Refer to Client Data Objects for more information.

The process of providing information about objects for the FDM Metadata is entitled Registration. The FDM Database Creation and Database Upgrade processes seed registration information for all FDM Reserved objects. The Registration process within the FDM Administration application is only used for registering client data objects.

Caution: Dropping, unregistering, or modifying FDM Reserved objects causes problems with the operation of OFS applications. This includes FDM Reserved tables and views, as well as the triggers, constraints, indexes, and public synonyms that support them.

Budgeting & Planning in the FDM Environment

Oracle Budgeting & Planning integrates with data from the Financial Data Manager database. However, it employs a different technology stack from FDM. Budgeting & Planning is runs in an Oracle Express, and Oracle Financial Analyzer environment instead of the Oracle RDBMS environment in which FDM exists. The concepts discussed in this chapter relating to object management pertain only to the Oracle RDBMS objects, not to Oracle Express objects.

Refer to Chapter 8, "Budgeting & Planning Server-Side Installation and Setup" and Chapter 9, "Budgeting & Planning Database Upgrade Process" for information on how to upgrade and maintain your Budgeting & Planning database.

Note: The Oracle Market Manager application is not included on the OFSA 4.5 CD. However, Market Manager Release 4.0 is compatible with the FDM 4.5 database (with the Market Manager database objects installed).

Oracle Market Manager in the FDM Environment

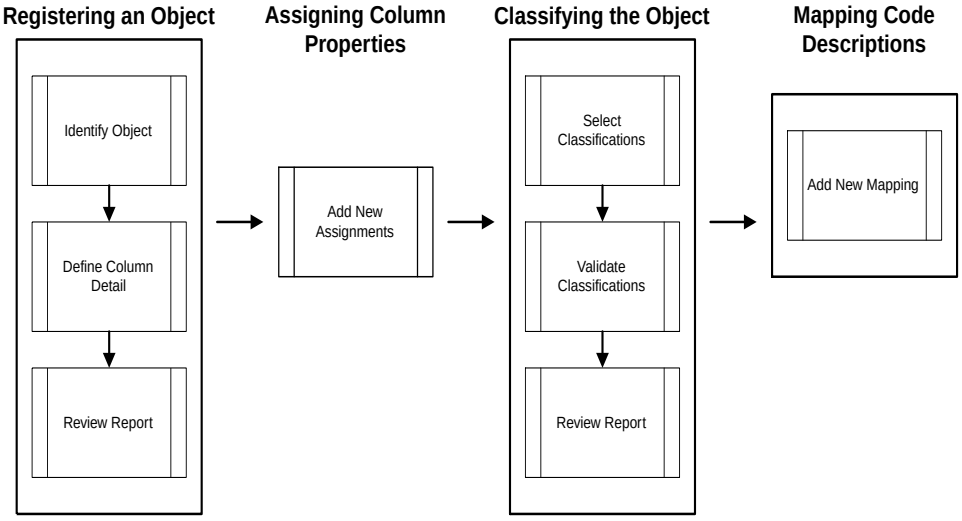
Market Manager is part of the Financial Data Manager database. All Market Manager objects are registered within the FDM Metadata. However, Market Manager does not employ the FDM Security Framework. In addition, Market Manager does not access or use the FDM Metadata.

Financial Data Manager is therefore *aware* of Market Manager database objects (because they are registered in the FDM Metadata), but the converse (Market Manager being *aware* of FDM) is not true. The Object Management information presented in this chapter is relevant for an Market Manager implementation only if FDM is implemented in conjunction with it. If you are running Market Manager in *Standalone* mode, none of the Object Management information present in this chapter has any bearing to your implementation.

Object Registration

FDM is an open environment supporting the use of customized tables and views. The process of identifying these customized objects in the FDM metadata is called Object Registration. The Object Registration process encompasses providing all of the metadata that FDM requires to use an object. FDM enables you to register any table or view owned by the FDM Schema Owner. In addition, you can register tables from other schemas, including schemas in another database instance.

The following is a pictorial representation of the Object Registration process:



The FDM metadata categories are:

- Object Identification
- Column Properties
- Table Classifications
- Table Properties
- Description Table Mapping

Note: (Note that Table Classifications are composed of Table Properties. You do not directly assign Table Properties to an Object, rather, you assign Table Classifications.

Each of these categories is described briefly. For detailed information regarding the Object Registration process within the FDM Administration application, refer to the *Oracle Financial Data Manager Administration Guide.*

Object Identification

Object identification is the first step of the Object Registration process. FDM stores identification information about an object in the following tables:

- OFSA_TABLES
- OFSA_TABLES_MLS
- OFSA_TAB_COLUMNS
- OFSA_TAB_COLUMNS_MLS

Identifying Objects from Other Schemas

When registering a table owned by a schema other than the FDM Schema Owner, FDM requires the existence of a view owned by the FDM Schema Owner to point to the table. The FDM Administration application then registers this view, rather than the actual table, in the FDM Metadata. If the table exists within the same database instance as the FDM Schema Owner, then the FDM Administration application automatically creates the view for that table. If the table exists in a separate database instance, you must create the view manually. The view then references the table using a database link to the other instance. Again, when registering a table from another database schema or instance, it is the view that is actually registered within the FDM Metadata, not the physical table.

Column Properties

Column Properties provide additional information about the nature of columns. These properties define how columns are used by the OFS applications. Column Property assignments for each column are stored in OFSA_COLUMN_ PROPERTIES.

The FDM Object Registration Wizard automatically populates default Column Property Assignments for any FDM Reserved Columns, except for the Processing Key Column Property. You, as the administrator, are responsible for assigning any Column Property assignments for user defined columns.

> **Note:** The Processing Key Column Property is required for the columns that compose the primary or unique key for each registered object.

For detailed information regarding FDM Column Properties, refer to the *Oracle Financial Data Manager Administration Guide*.

Table Classifications

Table Classifications provide a means to designate how tables and views are used within the OFS Applications. Each Table Classification identifies a specific purpose for which an assigned table or view is allowed to be used.

Some Table Classifications have requirements that must be satisfied in order for an object to be assigned to the classification. These requirements are designated by Table Properties associated to the Table Classifications. These Table Properties are either specific column name requirements or logic validations.

Table Classification assignments are stored in OFSA_TABLE_CLASS_ ASSIGNMENT.

Table Classifications are categorized as follows:

- User Assignable
- Reserved
- Dynamic

Note: FDM requires that all registered Client Data objects be assigned at least one Table Classification. FDM Reserved Objects are automatically assigned Table Classifications by the Database Upgrade and Database Creation processes.

User Assignable Table Classifications

User Assignable Table Classifications are those that can be assigned by the administrator to user-defined and client data objects, including the FDM Instrument tables installed with the database. These Table Classifications identify processing and reporting functions for the OFS applications. Some of these Table Classifications have requirements that must be met in order for the classification to be assigned to a table or view.

All User Assignable Table Classifications are available for assignment within the FDM Administration Table Classification Assignment Wizard. The following table lists the User Assignable Table Classifications:

Code	Table Classification Name
20	Instrument
100	Portfolio
200	TP Cash Flow
210	TP Non-Cash Flow
295	Codes User-Defined
296	MLS Descriptions User Defined
298	Code Descriptions User Defined
300	Transaction Profitability
310	Instrument Profitability
320	User Defined
330	Data Correction Processing
360	RM Standard
370	TP Option Costing

Note: For a detailed list of columns and logic requirements for each of the Table Classifications, refer to FDM Table Properties.

Validating User Assignable Table Classifications

FDM requires specific table structures, column names and column characteristics for OFS application operations. These structures and requirements are embodied by the User Assignable Table Classifications.

When you assign a Table Classification to a registered table or view in FDM Administration, the application validates that all requirements for that Table Classification are met. If all requirements are not met, the Table Classification assignment is rejected. You must then modify the object appropriately to meet the requirements for that particular Table Classification.

Each Table Classification comprises individual Table Properties that define the requirements for that classification. Table Properties are two distinct types: those

encompassing specific column requirements and those encompassing logic requirements via stored procedures.

Column Requirements

FDM implements column requirements by reserving specific column names. In order for an object to be assigned to a particular Table Classification, all of the FDM reserved column names required by that classification must exist on the object. In addition, the columns on the object must possess the required characteristics of those reserved column names. For example, the MATURITY_DATE column is required by the RM Standard Table Classification. In order to be assigned to this classification, the MATURITY_DATE column must exist on the object and it must be defined as data type DATE.

The FDM Table Classification Assignment Wizard identifies any missing columns for each Table Classification validation. If all columns are present on an object, but the object still fails the classification assignment, then one or more of the columns does not match the characteristic requirements. All column characteristics are specified in the OFSA_COLUMN_REQUIREMENTS table.

The OFSA_COLUMN_REQUIREMENTS table stores all of the required attributes for FDM reserved column names. Although it also stores attributes for any user defined Leaf Columns or any user-defined Portfolio columns, the Table Classification validation process accesses only this table for FDM Reserved columns when validating whether or not a column passes the requirements test.

FDM enables you to edit some of the fields in OFSA_COLUMN_REQUIREMENTS for the FDM Reserved columns (FDM Reserved columns are identified by PROTECTED_FLG=1). The User Edit? column designates whether or not the value for the column attribute can be changed. Columns marked with a User Edit? = NO are reserved by FDM and cannot be changed. You can always change any of the columns for user-defined entries in this table (identified by PROTECTED_FLG=0).

Note: You are not allowed to delete any records from OFSA_COLUMN_REQUIREMENTS where PROTECTED_FLG=1. Such columns are reserved by FDM for OFS application operations. You are allowed to insert new records into this table with PROTECTED_FLG=0 when creating new Portfolio columns.

Column Name	Description	User Edit?
COLUMN_NAME	Identifies the column for which requirements exist.	NO
OFSA_DATA_TYPE_CD	Designates the FDM Data Type requirement for the column. FDM Data Type Codes are stored in OFSA_DATA_ TYPE_DSC	NO
DATA_LENGTH	Designates the required Oracle data length	YES
DATA_PRECISION	Designates the required Oracle data precision.	YES
DATA_SCALE	Designates the required Oracle data scale	YES
DATA_TYPE	Designates the required Oracle data type.	NO
NULLABLE	Indicates if the column is nullable.	NO
DBF_NAME	Designates the name for the column when exported into a DBF.	YES
PROTECTED_FLG	A 1 indicates that the column_name is FDM Reserved. A 0 indicates that the column is user-defined.	NO

Use the following matrix to identify what to do when you fail a Table Classification validation for an object due to a column requirements discrepancy:

Type of Column	Failure Condition	Resolution
FDM Reserved	Data Length, Data Precision, Data Scale	Choice: Alter the column to match the requirement, or alter the requirement to match the column. Whichever is appropriate.
FDM Reserved	OFSA Data Type CD, Data Type,	Alter the column to match the requirement.
User Defined	Any requirement	Choice: Alter the column to match the requirement, or alter the requirement to match the column. Whichever is appropriate.

Note: If you modify the Data Length, Data Precision or Data Scale requirements for a column, all objects with that column must be modified to meet the new requirement. To do this, use the Utility script for altering Balance column definitions described in Chapter 21, "FDM Utilities". Do NOT modify the column requirement in OFSA_COLUMN_REQUIREMENTS unless you intend to modify all objects with that column.

Stored Procedure Requirements

Table Properties also validate specific logic tests. For example, the Instrument Profitability Table Classification requires an object with all of the registered Leaf Columns of type Both Instrument and Ledger. This requirement is implemented in FDM using a stored procedure validation. When you assign the Instrument Profitability Table Classification to an object, the Table Classification Assignment Wizard executes this stored procedure. If the stored procedure returns with a *True* result, the object meets the requirements and can be assigned to the classification. If the stored procedure returns with a *False* result, the Table Classification assignment is rejected.

The FDM Table Classification Assignment Wizard indicates whether or not an object passed all of the stored procedure validations for a Table Classification assignment. However, in order to identify which specific stored procedure requirement was not met, you must execute a procedure directly in SQL*Plus.

Running the Validation Check

FDM Administration validates that an object meets the table classification requirements during assignment. However, the message window that appears in FDM Administration after a Table Classification assignment does not display all of the detailed information as to why the object failed. The message window only identifies if a column is missing from the object or that one of the stored procedures failed.

To identify the specific reasons why an object failed a Table Classification assignment, run the validation procedure in SQL*Plus as follows:

```
<SQL> set serveroutput on
<SQL> execute ofsa_app_utils.check_table_class_reg(object_name, table_
classification_cd);
```

The *object_name* parameter is the name of the object to which you are assigning the Table Classification. The *table_classification_cd* parameter is the code number of the

Table Classification being validated for that object. For example, if you are assigning the Instrument Profitability classification to the DEPOSITS table, you would execute the following statement in SQL*Plus:

```
<SQL> set serveroutput on
execute ofsa_app_utils.check_table_class_reg('DEPOSITS',310);
```

The stored procedure then returns messages indicating why the classification assignment failed.

> **Note:** The instructions for manually executing the stored procedure validations are also included on the Wizard report page that appears after all Table Classification assignments have been validated.

Table Classification Validation Messages

The following messages identify the different reasons for Table Classification assignment failure:

Table Classification Message	Description
Column did not meet detail requirements	The column is registered for the object, but does not have the correct column attributes as specified in OFSA_COLUMN_REQUIREMENTS. Required attributes include: FDM Data Type (OFSA) Data Type (Oracle) Data Length Data Scale Precision
Missing column	The identified column is missing from the object
Invalid Join or Union View	FDM does not allow union or join views to be classified.
Invalid View	The view must be a valid view in the database.
No valid unique index found	The object does not meet the unique index requirements for the Table Classification.
Processing Key Column Properties do not match unique index	The columns in the unique index are not designated with the Processing Key Column Property. Assign this Column Property to the appropriate columns within FDM Administration.
Invalid Table	The table must be a valid object in the database.
Data Type must be NUMBER and FDM Data Type must be LEAF	Leaf Columns must be of Oracle Data Type NUMBER and be registered as FDM Data Type LEAF.

User Assignable Table Classification Descriptions and Requirements

All of the User Assignable Table Classifications are described with their requirements, as follows.

20 - Instrument

The Instrument classification identifies objects storing client account data. This Table Classification is a super-type grouping client data tables for reporting purposes. There are no requirements for this classification. In addition, because it is a super-type, it is automatically assigned to any table or view belonging to the following Table Classifications:

200 TP Cash Flow

210 TP Non-Cash Flow

300 Transaction Profitability

310 Instrument Profitability

330 Data Correction Processing

360 RM Standard

370 TP Option Costing

100 - Portfolio

The Portfolio classification identifies those objects that have a set of columns in common. Objects identified as 'Portfolio' are available for use with any Portfolio compatible OFSA ID. This enables users to create IDs that are not object specific. Rather, they are generic IDs where the object to be run against is provided at runtime. Assigning a table or view to the Portfolio classification enables the object to appear in the list of values for 'Portfolio tables' within the following OFS applications:

- Balance and Control
- Performance Analyzer
- Portfolio Analyzer
- Risk Manager
- Transfer Pricing

The Portfolio classification is the only Table Classification that is editable and controllable by the administrator. Administrators can add or subtract columns from the requirement list of this classification by specifying information in the

appropriate FDM metadata tables. Refer to Modifying the Portfolio Table Classification for information on how to modify the columns required for the Portfolio classification.

295 - Codes User-Defined

The 'Codes User Defined' classification identifies the object as a base table for storing user-defined code values. Code value base tables are those tables that store the list of allowable code values and do not store any translatable descriptions. The administrator assigns this Table Classification in those cases where they have added a new code column to the Financial Data Model. For example, a column 'TRANSACTION_CD' might designate numeric values for which there are translatable descriptions. The table storing the list of allowable Transaction codes is identified as Table Classification 295.

The tables supporting user-defined Code Columns can be multi-language enabled, but are not required to be. For a multi-language situation, there is a base table storing the allowable code values (295), an MLS table storing the translatable descriptions (296), and a language compatible view for displaying codes and their descriptions for reporting and viewing purposes (298). If the Code Descriptions are not multi-language enabled, then the table storing the both the base code values and the code descriptions should be classified as a '298 - Code Descriptions User Defined', instead of '295 - Codes User Defined' or '296 MLS Descriptions User Defined'.

In general, the Codes classification (295) is assigned to tables, not views. The exception to this rule is when the table storing the list of allowable code values is physically located in another schema. In this situation, assign the Codes classification to the view registered in the FDM schema.

296 - MLS Descriptions User Defined

The MLS (Multi-Language Support) Descriptions classification identifies the object as storing translatable descriptions for a code column. The administrator assigns this Table Classification in those cases where they have added a new code column to the Financial Data Model. For example, a column 'TRANSACTION_CD' might designate numeric values for which there are translatable descriptions. The table storing the list of translatable descriptions for Transaction codes is identified as Table Classification 196.

Only assign this classification to the table storing translatable descriptions in a multi-language environment. The tables supporting user-defined Code Columns can be multi-language enabled but are not required to be. For a multi-language situation, there is a base table storing the allowable code values (295), an MLS table

storing the translatable descriptions (296), and a language compatible view for displaying codes and their descriptions for reporting and viewing purposes (298). If the Code Descriptions are not multi-language enabled, then the table storing the both the base code values and the code descriptions should be classified as a '298 - Code Descriptions User Defined', instead of '295 - Codes User Defined' or '296 MLS Descriptions User Defined'.

In general, the MLS Descriptions classification (296) is assigned to tables, not views. The exception to this rule is when the table storing the list of allowable code values is physically located in another schema. In this situation, assign the MLS Descriptions classification to the view registered in the FDM schema.

298 - Code Descriptions User Defined

This classification identifies the view or table used for retrieving descriptions for a code column.

In a multi-language environment, this is separate from the table used to store the descriptions. In this situation, the object for displaying the code descriptions is actually a 'Language Compatible Views'. These views are dynamic in that they display descriptions only in the user's selected language, even when descriptions in multiple languages are stored in the MLS Description table.

In a single language environment, this classification is assigned to the table that stores both the code values and the code descriptions.

The tables supporting user-defined Code Columns can be multi-language enabled, but are not required to be. For a multi-language situation, there is a base table storing the allowable code values (295), an MLS table storing the translatable descriptions (296), and a language compatible view for displaying codes and their descriptions for reporting and viewing purposes (298). If the Code Descriptions are not multi-language enabled, then the table storing the both the base code values and the code descriptions should be classified as a '298 - Code Descriptions User Defined', instead of '295 - Codes User Defined' or '296 MLS Descriptions User Defined'.

The administrator assigns this Table Classification in those cases where they have added a new code column to the Financial Data Model. For example, a column 'TRANSACTION_CD' might designate numeric values for which there are translatable descriptions. The view for displaying translatable descriptions for Transaction codes would be assigned Table Classification 298.

200 - TP Cash Flow

The TP Cash Flow classification identifies objects for Transfer Pricing Cash Flow processing. Any table or view assigned to this Table Classification appears in the Transfer Pricing Process ID table list.

In order for a table or view to be classified for use with Transfer Pricing Cash Flow processing, the following Table Property requirements must be satisfied:

- Basic Instrument Requirements
- Cash Flow Proc. Requirements
- Cash Flow Edit Requirements
- Multi-Currency
- TP Basic Requirements
- Validate Instrument Leaves
- Validate Instrument Key

For detailed information about these Table Classification requirements, refer to FDM Table Properties.

210 - TP Non-Cash Flow

The TP Non-Cash Flow classification identifies objects for Transfer Pricing Non-Cash Flow processing. Any table or view assigned to this Table Classification appears in the Transfer Pricing Process ID table list.

In order for a table or view to be classified for use with Transfer Pricing Cash Flow processing, the following Table Property requirements must be satisfied:

- Basic Instrument Requirements
- Multi-Currency
- TP Basic Requirements
- Validate Instrument Leaves
- Validate Instrument Key

For detailed information about these Table Classification requirements, refer to FDM Table Properties.

300 - Transaction Profitability

The Transaction Profitability classification identifies objects for Performance Analyzer Allocation processing. Transaction Profitability objects store transaction data relating to accounts in Instrument Profitability objects. There is a many to one relationship between data in the Transaction Profitability objects to data in the Instrument Profitability objects. Any table or view assigned to this Table Classification appears in the Performance Analyzer Allocation ID table list.

In order for a table or view to be classified as a Transaction object for use with Allocation processing, the following Table Property requirements must be satisfied:

- Basic Instrument Requirements
- Multi-Currency
- Validate Instrument Leaves
- Validate Transaction Key

For detailed information about these Table Classification requirements, refer to FDM Table Properties.

310 - Instrument Profitability

The Instrument Profitability classification identifies objects for Performance Analyzer Allocation processing. Instrument Profitability objects store customer account data (for example, deposits, mortgages, investments). Any table or view assigned to this Table Classification appears in the Performance Analyzer Allocation ID table list.

In order for a table or view to be classified as an Instrument Profitability object for use with Allocation processing, the following Table Property requirements must be satisfied:

- Basic Instrument Requirements
- Multi-Currency
- Validate Instrument Leaves
- Validate Instrument Key

For detailed information about these Table Classification requirements, refer to FDM Table Properties.

320 - User Defined

The User-Defined classification has no requirements. This classification is only for grouping tables or views created by the administrator that are not assigned one of the other Table Classifications.

330 - Data Correction Processing

The Data Correction Processing classification identifies objects for use with Balance and Control Data Correction Processing. Tables or views assigned to this Table Classification appear in the Data Correction Processing ID table list.

In order for a table or view to be classified for use with Data Correction Processing, the following Table Property requirements must be satisfied:

- Validate Correction Key

For detailed information about this Table Classification requirements, refer toFDM Table Properties.

360 - RM Standard

The RM Standard classification identifies objects for Risk Manager processing. Any table or view assigned to this Table Classification appears in the Risk Manager Process ID table list.

In order for a table or view to be classified for use with Risk Manager processing, the following Table Property requirements must be satisfied:

- Basic Instrument Requirements
- Multi-Currency
- Cash Flow Proc. Requirements
- Validate Instrument Leaves
- Validate Instrument Key

For detailed information about these Table Classification requirements, refer to FDM Table Properties.

370 - TP Option Costing

The TP Option Costing classification identifies objects for Transfer Pricing Option Costing processing. Any table or view assigned to this Table Classification appears in the Transfer Pricing Process ID table list.

In order for a table or view to be classified for use with Transfer Pricing Option Costing processing, the following Table Property requirements must be satisfied:

- Basic Instrument Requirements
- Multi-Currency
- TP Basic Requirements
- TP Option Costing Requirements
- Validate Instrument Leaves
- Validate Instrument Key

For detailed information about these Table Classification requirements, refer to FDM Table Properties.

Modifying the Portfolio Table Classification

As stated, the only modifiable Table Classification is the Portfolio classification. This classification enables you to create IDs within the OFS applications that apply to the virtual Portfolio instrument object, rather than a specific database object. These IDs are then generic and can be applied against any object assigned to the Portfolio Table Classification.

The list of fields designated for the Portfolio classification is completely customizable. The FDM database creation process provides a seeded list of Portfolio fields. If you are migrating from OFSA Releases 3.5 or 4.0, the Database Upgrade Process converts all of the fields designated as Portfolio in your existing database to the new Portfolio Table Classification. You are allowed to add and remove fields from this list as needed.

The Portfolio Table Classification is composed of a single Table Property, also named Portfolio. The list of fields for this property is specified in the OFSA_PROPERTY_COLUMNS table where the table_property_cd = 40.

To identify a user defined column as a new Portfolio column, complete the following:

1. Column Requirements Insert

   Create a row in the OFSA_COLUMN_REQUIREMENTS table specifying the required attributes for the Portfolio field. The OFS applications reference these attributes for any occurrence of the Portfolio column in a Portfolio ID.

Column Name	Description
COLUMN_NAME	Identifies the column name for the new Portfolio column
OFSA_DATA_TYPE_CD	Designates the FDM Data Type requirement for the column. FDM Data Type Codes are listed in OFSA_DATA_ TYPE_DSC
DATA_LENGTH	Designates the Oracle data length for the Portfolio column.
DATA_PRECISION	Designates the Oracle data precision for the Portfolio column.
DATA_SCALE	Designates the Oracle data scale for the Portfolio column.
DATA_TYPE	Designates the Oracle data type for the Portfolio column.
NULLABLE	Indicates if the column is nullable.
DBF_NAME	Designates the name for the column when exported into a DBF.
PROTECTED_FLG	A 0 indicates that the column is user-defined. A 1 indicates that the column_name is FDM Reserved. Only insert values with 0 to designate user-defined fields.

2. Table Property Insert

Insert a row into OFSA_PROPERTY_COLUMNS as follows:

```
INSERT INTO ofsa_property_columns
VALUES (40, :column_name, 0);
```

Caution: When modifying the list of Portfolio fields in OFSA_PROPERTY_COLUMNS, you must insert or delete only records for table_property_cd=40. If you delete any records for other Table Property Codes, you invalidate the Table Classification Assignment process.

FDM Reserved Table Classifications

Reserved Table Classifications are assigned to tables and views by the FDM Database Creation and Database Upgrade processes. These Table Classifications are used internally by FDM processes and other OFSA processes. Reserved Table Classifications are not available for assignment by administrators.

The following Table Classifications are designated as FDM reserved:

Code	Table Classification Name
10	OFSA ID
30	Detail Leaf Information
40	Leaf Results
50	Ledger Stat
60	Customer Householding
70	FDM Reserved
80	Customer Householding Processing
110	Processing Audit
130	Results for Reporting
140	RM Detail Results
150	RM Leaf Results
160	RM Results Template
165	RM EAR Template
180	RM VAR Views

Code	Table Classification Name
182	RM VAR Results
185	RM EAR Results
190	FDM System
195	Codes (FDM Reserved)
196	MLS Descriptions (FDM Reserved)
197	Code Descriptions (FDM Reserved)
198	Code Descriptions (User Editable)
220	Transformed Ledger Stat
230	Transformed RM Cash Flow
240	Transformed RM GAP
250	Transformed Tree Rollup
260	Trans Ledger Stat Template
270	Trans RM Cash Flow Template
280	Trans RM GAP Template
290	Trans Rollup Template
350	Primary Key All Leaves
351	All B Leaves
352	All L Leaves
420	Transformed RM Result Detail
430	Transformed RM Cons Cash Flow
440	Transformed RM Cons GAP
450	Reporting System Tables
460	Reporting Rates
470	Business Process Audit
480	Data Verification View/Update
490	Collateral Objects

Dynamic Table Classifications

Dynamic Table Classifications are assigned to output tables created dynamically as a result of Risk Manager or Transformation processing. Because these Table Classifications are assigned automatically by Risk Manager and Transformation Processing, they are not assignable by users in the Table Classification Assignment Wizard in the FDM Administration application. However, because administrators need to be able to grant privileges for dynamic objects, these Table Classifications are available for grant assignments in the FDM Dynamic Table Classification Privileges tab within the FDM Administration application.

The following Table Classifications are designated as Dynamic:

Code	Table Classification Name
140	RM Detail Results
185	RM EAR Results
220	Transformed Ledger
230	Transformed RM Cash Flow
240	Transformed RM GAP
250	Transformed Tree Rollup

For information on how to assign Dynamic Object Privileges, refer to the *Oracle Financial Data Manager Administration Guide.*

FDM Table Properties

Table properties identify required characteristics for table classifications. In order for an object to receive a particular classification, it must meet the requirements specified by the table properties (if any) of that classification.

There are two types of table properties: column names and stored procedures.

Column Name Table Properties

Column name table properties consist of required column names, as well as required column characteristics, for table classifications. In order for an object to be assigned to a particular table classification, the object must meet all of the requirements for column names and column characteristics of the table properties of that table classification. For example, several classifications require the existence of an

AS_OF_DATE column, which is of data type *DATE* on the table or view, in order for the object to be assigned that classification.

These column requirements are validated whenever a table or view is assigned to a new table classification within the Table Classification Assignment Wizard of FDM Administration.

The following table properties identify column name requirements:

Code	Description
10	Basic Instrument Requirements
40	Portfolio Requirements
50	Cash Flow Proc. Requirements
60	Cash Flow Edit Requirements
80	Multi-Currency Requirements
100	TP Option Costing Requirements
110	TP Basic Requirements

10 - Basic Instrument Requirements This table property identifies a set of columns that are required by the financial instrument processing functions, such as Allocation processing, Risk Manager processing, and Transfer Pricing processing. The required columns are:

ID_NUMBER

IDENTITY_CODE

AS_OF_DATE

IDENTITY_CODE_CHG

40 - Portfolio Requirements This table property identifies a set of common columns for objects assigned to the Portfolio table classification. The Portfolio table classification enables users to create OFSA IDs that are generic and not specific to a particular object. Such IDs are then usable with any object having all of the fields specified by the Portfolio table property.

The FDM database creation process provides a default list of columns for the Portfolio table property. The administrator can add or remove columns from this list. The Portfolio table property is the only one modifiable by the administrator. The proce-

dure for adding and removing Portfolio columns is detailed in the description of the Portfolio Table Classification.

The following is the default list of Portfolio fields provided by the FDM database creation process:

ACCOUNT_OFFICER

ACCRUAL_BASIS_CD

ADJUSTABLE_TYPE_CD

AS_OF_DATE

AVG_BOOK_BAL

AVG_NET_BOOK_BAL_C

BANK_CODE

BRANCH_CODE

COMMON_COA_ID

COMPOUND_BASIS_CD

CUR_BOOK_BAL

CUR_GROSS_RATE

CUR_NET_BOOK_BAL_C

CUR_NET_PAR_BAL_C

CUR_NET_RATE

CUR_PAR_BAL

CUR_TP_PER_ADB

CUR_YIELD

DATA_SOURCE

DEFERRED_ORG_BAL

GEOGRAPHIC_LOC_CD

GL_ACCOUNT_ID

IDENTITY_CODE

IDENTITY_CODE_CHG

ID_NUMBER
INSTRUMENT_TYPE_CD
INTEREST_RATE_CD
ISO_CURRENCY_CD
LAST_REPRICE_DATE
LAST_UPDATE_DATE_C
MARGIN
MARGIN_GROSS
MARGIN_T_RATE
MARKET_SEGMENT_CD
MARKET_VALUE_C
MATCHED_SPREAD_C
MATURITY_DATE
NEG_AMRT_AMT
NEG_AMRT_EQ_DATE
NEG_AMRT_EQ_FREQ
NEG_AMRT_EQ_MULT
NEG_AMRT_LIMIT
NET_MARGIN_CD
NEXT_PAYMENT_DATE
NEXT_REPRICE_DATE
ORG_BOOK_BAL
ORG_NET_BOOK_BAL_C
ORG_NET_PAR_BAL_C
ORG_PAR_BAL
ORG_RATE
ORG_UNIT_ID
ORIGINATION_DATE

PERCENT_SOLD

PMT_ADJUST_DATE

PMT_CHG_FREQ

PMT_CHG_FREQ_MULT

PMT_DECR_CYCLE

PMT_DECR_LIFE

PMT_FREQ

PMT_FREQ_MULT

PMT_INCR_CYCLE

PMT_INCR_LIFE

PRIOR_TP_PER_ADB

PRODUCT_TYPE_CD

RATE_CAP_LIFE

RATE_CHG_MIN

RATE_CHG_RND_CD

RATE_CHG_RND_FAC

RATE_DECR_YEAR

RATE_FLOOR_LIFE

RATE_INCR_YEAR

RECORD_COUNT

REMAIN_TERM_C

REMAIN_TERM_MULT_C

REPRICE_FREQ

REPRICE_FREQ_MULT

SIC_CD

TAX_EXEMPT_PCT

TEASER_END_DATE

TRANSFER_RATE

TRAN_RATE_REM_TERM

T_RATE_INT_RATE_CD

50 - Cash Flow Proc. Requirements This table property identifies the columns required for cash flow processing operations in Risk Manager and Transfer Pricing. This property is required for the following table classifications:

Code	Description
200	TP Cash Flow
360	RM Standard
370	TP Option Costing

The Cash Flow Processing Requirements table property is composed of the following fields:

ACCRUAL_BASIS_CD

ADJUSTABLE_TYPE_CD

AMRT_TERM

AMRT_TERM_MULT

AMRT_TYPE_CD

AS_OF_DATE

COMPOUND_BASIS_CD

CUR_BOOK_BAL

CUR_GROSS_RATE

CUR_NET_RATE

CUR_PAR_BAL

CUR_PAYMENT

CUR_TP_PER_ADB

DEFERRED_ORG_BAL

IDENTITY_CODE

ID_NUMBER

INSTRUMENT_TYPE_CD

INTEREST_RATE_CD

INT_TYPE

ISSUE_DATE

LAST_PAYMENT_DATE

LAST_REPRICE_DATE

LRD_BALANCE

MARGIN

MARGIN_GROSS

MARGIN_T_RATE

MARKET_VALUE_C

MATCHED_SPREAD_C

MATURITY_DATE

NEG_AMRT_AMT

NEG_AMRT_EQ_DATE

NEG_AMRT_EQ_FREQ

NEG_AMRT_LIMIT

NET_MARGIN_CD

NEXT_PAYMENT_DATE

NEXT_REPRICE_DATE

ORG_PAR_BAL

ORG_PAYMENT_AMT

ORG_TERM

ORG_TERM_MULT

ORIGINATION_DATE

PERCENT_SOLD

PMT_ADJUST_DATE

PMT_CHG_FREQ

PMT_CHG_FREQ_MULT

PMT_DECR_CYCLE

PMT_DECR_LIFE

PMT_FREQ

PMT_FREQ_MULT

PMT_INCR_CYCLE

PMT_INCR_LIFE

PRIOR_TP_PER_ADB

RATE_CAP_LIFE

RATE_CHG_MIN

RATE_CHG_RND_CD

RATE_CHG_RND_FAC

RATE_DECR_CYCLE

RATE_FLOOR_LIFE

RATE_INCR_CYCLE

RATE_SET_LAG

RATE_SET_LAG_MULT

REMAIN_NO_PMTS_C

REPRICE_FREQ

REPRICE_FREQ_MULT

TEASER_END_DATE

TRANSFER_RATE

TRAN_RATE_REM_TERM

T_RATE_INT_RATE_CD

60 - Cash Flow Edit Requirements This table property identifies the columns required by the cash flow edit process, which are in addition to the Cash Flow Processing Requirements table property. The cash flow edit process is a Data Correction Processing function provided with Balance and Control for data scrubbing in prepara-

tion of running a cash flow process. This property is required for the following table classifications:

Code	Description
200	TP Cash Flow
360	RM Standard
370	TP Option Costing

The Cash Flow Edit Requirements table property is composed of the following fields:

CUR_NET_PAR_BAL_C

ORG_BOOK_BAL

REMAIN_TERM_MULT_C

80 - Multi-Currency Requirements This table property identifies the column that is required for processing functionality that is multi-currency enabled. The following table classifications require this property:

Code	Description
200	TP Cash Flow
210	TP Non-Cash Flow
300	Transaction Profitability
310	Instrument Profitability
360	RM Standard
370	TP Option Costing

The Multi-Currency Requirements table property is composed of the following field:

ISO_CURRENCY_CD

100 - TP Option Costing Requirements This table property identifies columns required for running Transfer Pricing option costing processing on a table or view. Only the TP Option Costing table classification (370) requires this property.

The fields required for TP Option Costing are:

CUR_OAS

CUR_STATIC_SPREAD

HISTORIC_OAS

HISTORIC_STATIC_SPREAD

ORG_MARKET_VALUE

110 - TP Basic Requirements This table property identifies columns required for running basic Transfer Pricing processing on a table or view. Each of the Transfer Pricing processing table classifications require this property:

Code	Description
200	TP Cash Flow
210	TP Non-Cash Flow
370	TP Option Costing

The fields required for Transfer Pricing processing are:

ADJUSTABLE_TYPE_CD

CUR_NET_RATE

CUR_TP_PER_ADB

INTEREST_RATE_CD

LAST_REPRICE_DATE

MARGIN

MATCHED_SPREAD_C

MATURITY_DATE

NEXT_REPRICE_DATE

ORIGINATION_DATE

PERCENT_SOLD

PRIOR_TP_PER_ADB

RATE_CAP_LIFE

RATE_CHG_MIN

RATE_CHG_RND_CD

RATE_CHG_RND_FAC

RATE_FLOOR_LIFE

REPRICE_FREQ

REPRICE_FREQ_MULT

TEASER_END_DATE

TRANSFER_RATE

TRAN_RATE_REM_TERM

Stored Procedure Table Properties

Stored Procedure table properties consist of validation logic required for table classifications. For an object to be assigned to a particular table classification, it must meet all of the logic requirements for the classification, if any. For example, the RM Standard Table Classification requires that any "B" ("Both Instrument and Ledger") leaf columns on an object be assigned to that classification.

Logic validations are implemented using Oracle stored procedures. These stored procedures are run whenever an object is assigned to a new table classification in the Table Classification Assignment Wizard of FDM Administration.

The following table properties identify logic validation requirements:

Code	Description
1000	Validate Instrument Leaves
1010	Validate Instrument Key
1020	Validate Transaction Key
1030	Validate Correction Key

1000 - Validate Instrument Leaves This procedure validates that all instrument leaf columns defined in the FDM metadata exist on the object. The FDM Leaves tab in FDM Administration displays all of the leaf columns defined in the database. The instrument leaf columns are those entries of *Type = B* ("Both Instrument and Ledger"). The Validate Instrument Leaves table property verifies that all of these leaf Col-

umns exist on the object being classified. This property is a requirement for the following table classifications:

Code	Description
200	TP Cash Flow
210	TP Non-Cash Flow
300	Transaction Profitability
310	Instrument Profitability
360	RM Standard
370	TP Option Costing

1010 - Validate Instrument Key This procedure validates that the unique key for a table or view is composed of the ID_NUMBER and IDENTITY_CODE columns. In addition, the procedure validates that the ID_NUMBER and IDENTITY_CODE columns on the object are assigned the Processing Key column property in the Column Property Assignment Wizard of FDM Administration.

If the object being validated is a view referencing another table, the validation logic checks the unique key of the physical table. Registration of views referencing other views is not allowed and such objects fail the Validate Instrument Key procedure. This property is a requirement for the following table classification:

Code	Description
200	TP Cash Flow
210	TP Non-Cash Flow
310	Instrument Profitability
360	RM Standard
370	TP Option Costing

1020 - Validate Transaction Key This procedure validates that the unique key for an object is composed of the ID_NUMBER, IDENTITY_CODE, and one or more B (“Both Instrument and Ledger”) leaf columns. In addition, the procedure validates that these columns on the object are assigned the Processing Key column property in the Column Property Assignment Wizard of FDM Administration.

If the object being validated is a view referencing another table, the validation logic checks the unique key of the physical table. Registration of views referencing other

views is not allowed and such objects fail the Validate Transaction Key procedure. This property is a requirement for the following table classifications:

Code	Description
300	Transaction Profitability

1030 - Validate Correction Key This procedure validates that a unique key for an object exists. In addition, the procedure validates that the columns in the unique key are assigned the Processing Key column property in the Column Property Assignment Wizard of FDM Administration.

If the object being validated is a view referencing another table, the validation logic checks the unique key of the physical table. Registration of views referencing other views is not allowed and such objects fail the Validate Correction Key procedure. This property is a requirement for the following table classification:

Code	Description
330	Data Correction Processing

Description Table Mapping

Description Table Mapping provides the means to designate the data structures where descriptions are stored for code value columns. The mapping of a Code column name to a data structure storing descriptions for that code is then accessed by the OFS applications for reporting and user-interface operations.

For example, the DEPOSITS table has a code column named ACCRUAL_BASIS_CD. This column stores values such as 1, 2, or 3. The real world descriptions for these code values are 30/360, Actual/360, and Actual/Actual. For the user to be able to view these descriptions in a report, or within the Data Filter ID user-interface, for example, the ACCRUAL_BASIS_CD column must be mapped to the appropriate table or view from which these descriptions are retrieved.

For MLS enabled Code Descriptions, map to the appropriate Language Compatible View supporting that Code Column. For non-MLS enabled Code Descriptions, map to the table storing the code names and descriptions.

Refer to Chapter 15, "FDM Multi-Language Support" for information on how to create the data structures to support MLS enabled Code Descriptions.

Client Data Objects

As stated earlier, FDM supports the use of customized database objects. Such objects are termed as Client Data Objects to distinguish them from FDM Reserved objects seeded in the database by the FDM Database Upgrade and Database Creation processes. The FDM actually creates default client data objects for financial instruments. Although these tables are created by the FDM Database Creation process, they are considered client data objects because FDM does not reserve the object names. Any table or view (except for the LEDGER_STAT table) that does not have an OFSA_ prefix is considered to be a client data object.

This section discusses the different types of Client Data Objects and how to manage them.

There are several distinct categories of client data objects:

- Instrument and Account
- User Defined Code Descriptions
- LEDGER_STAT
- Free Form

Instrument and Account

Instrument and Account objects are tables and views storing traditional financial services information about customers and accounts. These are the most commonly used objects for OFS processing and reporting operations.

Included in this group are all of the Instrument and Services tables created by the FDM Database Creation Process (referred to as seeded Instrument or seeded Services tables) as well as any other such objects created by users. You can customize these tables as needed for your implementation. You are allowed to unregister (and then drop) any of the seeded Instrument tables.

In order to be available for OFS processing operations, Instrument and Account must be properly classified with the FDM Table Classifications. Refer to Object Registration for information regarding the different Table Classifications available within FDM.

Creating New Instrument and Account Tables

Because most Instrument and Account tables are used for OFS processing operations, it is recommended that you use the Instrument Creation Template scripts provided with the database package when creating your own. These

templates provide the basic fields and indexes required for such tables to be used for OFS processing operations. You can edit these template scripts as needed for your implementation.

The template scripts are located in the following directory in the database package:

```
<UPGRADE_HOME>/utilities/instrument_templates
```

The templates provide instructions on how to use them and modify them for creating customized instrument and account tables. Once you have created your new tables, you need to register them for FDM using the Object Registration functionality within FDM.

Using Views

FDM supports the use of database views. This means that you can create views that serve as Instrument and Account objects for OFS processing and reporting functions. The OFS applications do not make a distinction between tables and views for any processing or reporting operations.

FDM does impose some limitations on the use of views for these purposes. These limitations are:

1. All views must be owned by the FDM Schema Owner. In order to be able to register a view with the FDM Metadata, the view must be owned by the FDM Schema Owner.
2. The FDM Table Classifications for OFS processing prohibit views on views. This means that you can create views to reference other views only if you do not intend on assigning one of the following FDM *processing* classifications:
   - TP Cash Flow (200)
   - TP Non-Cash Flow (210)
   - Transaction Profitability (300)
   - Instrument Profitability (310)
   - RM Standard (360)
   - TP Option Costing (370)

The FDM Administration application enables you to register and assign privileges for views just like tables.

Registering Tables in other Schemas

FDM enables you to register client data objects from schemas other than the FDM Schema Owner. When such an object is registered using the FDM Administration application, the application creates a view owned by the FDM Schema Owner to reference the newly registered object.

To register a table from another database instance, create a view on that table owned by the FDM Schema Owner referencing a database link to the instance. Then register the view in FDM Administration.

Seeded Instrument and Account Tables

The FDM Database Creation process creates the following default instrument and account objects. You can unregister or modify the objects as appropriate:

Instrument Tables

- COMMERCIAL_LOAN
- CONSUMER_LOAN
- CREDIT_CARDS
- DEPOSITS
- INVESTMENTS
- MORTGAGES
- MORTGAGE_BACK_SEC
- WHOLESALE_FUNDING
- TERM_DEPOSITS
- FORWARD_CONTRACTS
- INTEREST_RATE_OPTIONS
- INTEREST_RATE_SWAPS

Transaction Tables

- COMMERCIAL_LOAN_TRANSACTIONS
- CONSUMER_LOAN_TRANSACTIONS
- CREDIT_CARDS_TRANSACTIONS
- CREDIT_CARDS_TRANSACTIONS

- DEPOSITS_TRANSACTIONS
- INVESTMENTS_TRANSACTIONS
- MORTGAGES_TRANSACTIONS
- MORTGAGE_BACK_SEC_TRANSACTIONS
- WHOLESALE_FUNDING_TRANSACTIONS
- TERM_DEPOSITS_TRANSACTIONS

Services Tables

- CC
- CD
- CK
- CL
- CN
- IL
- HH
- DC
- IV
- LS
- MC
- ML
- OD
- OL
- OS
- RA
- SD
- SV
- TR

Customer and Account Tables

- ACCT
- CUST_ADDR
- CUST
- IND
- BUS

Collateral Tables

- ACCOUNT_COLLATERAL
- ACCOUNT_GUARANTOR_RELATION
- COLLATERAL_BOATS
- COLLATERAL_OWNERS
- COLLATERAL_REAL_ESTATE
- COLLATERAL_VEHICLES
- COLLATERAL
- COLLATERAL_ASSESSMENT_HISTORY
- COLLATERAL_AUCTION_DETAILS
- COLLATERAL_INSURANCE_DETAILS
- COLLATERAL_OTHER_INSTITUTIONS
- COLLATERAL_SHARES

User-Defined Code Descriptions

FDM provides MLS enabled data structures to support the code columns existing on the seeded Instrument and Account tables. However, if you add any new code columns to any of your Instrument and Account tables, you need to create and register the supporting data structures for the code descriptions. Such data structures are referred to as User-Defined Code Description objects.

Single Language Environment

In a single language environment, all that is needed is a single table providing the code descriptions. The table needs to have the code column itself, as well as one or more columns providing description information for the code values, such as a

short description and a long description. The FDM naming convention for such tables is the name of the Code column with an _DSC suffix.

For an example, assume that you have created a new code column ACCOUNT_ OFFICER_TYPE_CD and registered it for one or more of your instrument tables. The table to provide the short and long description values for this code column might be named ACCOUNT_OFFICER_TYPE_DSC and appear as follows:

Table Name: ACCOUNT_OFFICER_TYPE_DSC

Column Name	Column Definition
ACCOUNT_OFFICER_TYPE_CD	NUMBER(5)
ACCOUNT_OFFICER_TYPE	VARCHAR2(40)
DESCRIPTION	VARCHAR2(255)

After the table is created, register it for FDM using the FDM Administration application and assign it to an appropriate Table Classification. Appropriate Table Classifications for User Defined Code Description objects in a single language environment are as follows:

Object	Table Classification Assignment
Code Description Table	298 - Code Descriptions User-Defined

Complete the appropriate Description Table Mappings for any instrument tables on which the code column exists.

Multi-Language Environment

In a multiple language environment, you need to separate the translatable information (that is, the code descriptions) from the base table storing the code values. To do this, refer to Chapter 15, "FDM Multi-Language Support".

Managing Data for User Defined Code Descriptions

User reporting operations join data from Instrument and Account objects with the data in Code Description objects. If unique code values are missing from the mapped Code Description object for that code column, users report return fewer records than they should because the join excludes any records that do not match.

Therefore, it is critical that all unique code values for a given code column are correctly represented in the mapped Code Description object.

FDM provides a utility procedure Synchronize Instrument to automatically generate rows in the User-Defined Code Description data structures for any new code values in the Instrument and Account objects. This procedure reads the Description Table Mappings for each code column in a given Instrument and Account object to determine how to insert the missing rows. For more information on how to run this procedure, refer to Chapter 21, "FDM Utilities".

Note: The SYNCHRONIZE_INSTRUMENT procedure only inserts missing rows for User Defined and User Editable Code Descriptions objects. The procedure provides an error message for any code values in the Instrument and Account objects that do not exist in FDM Reserved Code Description objects that are protected from update.

LEDGER_STAT

The LEDGER_STAT table is an FDM Reserved Object Name that stores client data. FDM reserves this table name and the structure of this table for internal use. However, because the LEDGER_STAT table does store client account information, you are allowed to add new columns to the table (such as Leaf Columns).

If this table is dropped or unregistered from the FDM database, the Database Upgrade Process recreates it and its supporting metadata.

Loading Data into LEDGER_STAT

FDM provides the Ledger Stat Load procedure for loading data into the LEDGER_STAT table. For information on how to use this procedure, refer to Chapter 21, "FDM Utilities".

Maximum Number of Leaves on LEDGER_STAT

The maximum number of user-defined Leaf Columns allowed on the LEDGER_STAT table is 7. This number is in addition to the 4 FDM seeded Leaf Columns.

Free Form

Free Form objects include any user created tables and views that are not categorized as Instrument or Account objects. When creating such objects for FDM, assign the User Defined Table Classification to them during Object registration.

If you intend on running the Balance and Control Data Correction Processing functionality for Free Form objects, refer to the requirements for that Table Classification in Object Registration. Otherwise, because Free Form objects are not used for any OFS processing operations, there are no requirements or limitations on their definition.

Risk Manager Results Tables

Risk Manager results tables are created during processing of the Risk Manager Process ID to store output of the process. These tables tend to accumulate in the database over time as users create and run new Process IDs.

Types of Results Tables

There are two different categories of Risk Manager Results tables created during processing. These categories are:

- Scenario Based Results Tables
- Earnings at Risk Results Tables

Scenario Based Results Tables

Scenario Based Risk Manager processing outputs a portion of the results into static tables that store results for all such processes. However, this processing also creates new tables (referred to as dynamic objects) to store the detail results output.

Tables created for the detail results are created and named dynamically by the Risk Manager processing engine. These tables are created based upon a template table definition.

Each Scenario based Risk Manager process populates and/or creates the following output tables:

Type of Result Table	Table Name	Template Table Name
Static	OFSA_RESULT_MASTER	N/A
Static	OFSA_CONSOLIDATED_MASTER	N/A
Dynamic	RES_DTL_XXXXXX	OFSA_IDT_RESULT_DETAIL
Dynamic	CONS_DTL_XXXXXX	OFSA_IDT_RESULT_DETAIL

The Table Name format for all dynamic tables contains a number extension (designated by the XXXXXX in the list). The number extension is the SYS_ID_NUM of the Risk Manager Process ID that created the table.

For example, the table RES_DTL_123456 was created by the Risk Manager Process ID identified as SYS_ID_NUM=123456 in the OFSA_CATALOG_OF_IDS table.

Earnings at Risk Results Tables

Earnings at Risk (EAR) processing in Risk Manager creates dynamic tables to store process results. Tables created for the EAR results are created and named dynamically by the Risk Manager processing engine. These tables are created based upon a template table definition.

Each EAR process creates and populates the following output tables:

Type of Result Table	Table Name	Template Table Name
Dynamic	EAR_LEAF_DTL_XXXXXX	OFSA_EAR_LEAF_DTL
Dynamic	EAR_LEAF_AVG_XXXXXX	OFSA_EAR_LEAF_AVG
Dynamic	EAR_TOTAL_AVG_XXXXXX	OFSA_EAR_TOTAL_AVG
Dynamic	EAR_TOTAL_DTL_XXXXXX	OFSA_EAR_TOTAL_DTL

Dynamic Results Table Definition

The definition and structure of the dynamic Risk Manager results tables are based on template tables in the database. Risk Manager uses the structure and definition of the appropriate template table to create the results table during the execution of the Process ID.

When a Risk Manager Process ID is executed for the first time, the specification for the dynamic results output table, including column definitions and sizing parameters, is based upon the template table. During subsequent execution of the same Process ID, if either the dynamic results table or the template table has been modified so that the table structure and column definitions are no longer the same, Risk Manager drops and re-creates the results table using the template table as a basis. The drop and re-creation of the results table only occurs when the column order, column names, or column definitions in the results table do not match those of the template table. Differences in table sizing parameters do not cause the drop and re-creation of the results table.

The structure and definition of the static results tables is not affected by the Process ID. The Process ID populates these tables with data, but does not change their structure.

When the Risk Manager Process ID and the associated results are no longer needed, the Process ID should be deleted because the result tables can be very large. To delete the Risk Manager Result Detail table, delete the actual Process ID in Risk Manager. The Result Detail table is deleted when the Process ID is deleted.

For more information on the Results tables, refer to the *Oracle Risk Manager Reference Guide*

Transformation Output Tables

Users create transformation output tables by running the Transformation ID within the OFS applications. These tables are categorized as *dynamic objects* as they are created automatically by the OFS Transformation processing engine. FDM provides administrators with the ability to control output table definitions and storage parameters.

The following tasks related to output table and index creation are explained in this section:

- How the OFSA Leaf Setup is used to define column names for transformation output tables and rate weighting.
- How you can control which indexes are created for each Transformation ID output table.
- How you can control the physical storage parameters used to create each type of Transformation ID output table and its indexes.

Leaf Setup and Output Tables

The Leaf Setup for Financial Elements has three attributes: Column Name, Display Name and Weighting Financial Element. The Transformation ID uses Column Name and Display Name to define the Financial Element columns in the transformation output table.

A separate column is created in the transformation output table for each distinct Financial Element in the source table, or for a subset of these Financial Elements, as selected by the Data Filter associated with the Transformation ID. Note that the Transformation ID creates every transformation output table under the schema of the FDM Schema Owner, even though the ID is run by an end user.

The value for Column Name in Leaf Setup for each Financial Element is used as the column name in the output table for that Financial Element column. The Display Name for the Financial Element is used as its Display Name in the FDM Metadata, which defines the name that is displayed for each column inside the OFS applications.

> **Note:** Because of the way the Transformation ID uses Column Name and Display name, it is essential that you provide meaningful values for these attributes for all of your Financial Elements. The values for Column Name and Display Name in the Leaf Setup for all Financial Elements must be non-null and unique. The values for Column Name must also be in upper case.

The FDM database upgrade provides default values for these new columns. The Synchronize Instrument utility also provides defaults for the Column Name column. After the upgrade, and after each time you run the Synchronize Instrument utility, you should edit Leaf Setup to make sure that every Financial Element has a correct and meaningful value for each of these attributes, and that they are unique.

In the event that a Transformation ID encounters an invalid or duplicated Column Name value in Leaf Setup for the set of Financial Elements to be transformed, it stops processing and logs an error in the PROCESS_ERRORS table.

For each rate Financial Element column in the newly created transformation output table, the name of the weighting financial element column is recorded as a Related Field column property. The column name comes from the COLUMN_NAME column of the row in OFSA_DETAIL_ELEM whose LEAF_NODE value matches the WEIGHTING_FE value for the column in the output table. The column names recorded as the Related Field column property reflect the weighting financial element assignments that are in effect in the OFSA_DETAIL_ELEM table at the time the transformation output table is created.

Template Tables and Indexes

There are four kinds of transformations processed by the Transformation ID.

- Ledger_ Stat
- Risk Manager Result Detail Cash Flow (and Consolidated Cash Flow)
- Risk Manager Result Detail Gap (and Consolidated Gap)
- Tree Rollup

Consolidated Cash Flow and *Consolidated Gap* refer to output tables created by Risk Manager for results aggregated into the Reporting Currency.

Each of these transformations use a template table to define the columns in all of the output tables of that transformation type. The Transformation ID uses definitions in FDM Metadata for the template tables. These definitions are used to define FDM Metadata entries for columns in the output table that have no matching column in the source table.

The template tables are only definitions. They contain no data.

Each template table is recorded in the FDM Metadata with a Table Classification assignment that is unique to that template table.

The names of the template tables and associated Table Classification assignments appear in the following table.

Template Table Name	Table Classification
OFSA_TRANSFORM_LS_TEMPLATE	260 - Trans Ledger Stat Template
OFSA_TRANSFORM_RMC_TEMPLATE	270 - Trans RM Cash Flow Template
OFSA_TRANSFORM_RMG_TEMPLATE	280 - Trans RM GAP Template
OFSA_TRANSFORM_ROLLUP_ TEMPLATE	290 - Trans Rollup Template

The Table Classifications assigned to the template tables are FDM Reserved classifications. These classifications are not available for assignment to any other objects.

Template Indexes

The Ledger_Stat and the two Risk Manager templates are similar in definition and usage. For each of these three templates, any index on the template table is created on the Transformation ID output table at the end of the ID processing.

A single, unique index is pre-created by the FDM database upgrade on each of these three template tables. You can create additional indexes on these template tables if needed. Each additional index that you create on any of the template tables is created on each output table subsequently created using that template.

Naming Restrictions

Oracle index names must be unique within each schema. In order to guarantee the uniqueness of index names for indexes created on the output tables, the index name must contain the name of the output table. For the name of the index created on the output table, the portion of the template index name that matches the template table name is replaced with the output table name.

For example, if you run a Transformation ID of the Ledger Stat type with output table name MY_LS_TRANSFORMATION and the TRANSFORM_LS_TEMPLATE table has template indexes called:

- TRANSFORM_LS_TEMPLATE
- TRANSFORM_LS_TEMPLATE_1

then the indexes created on MY_LS_TRANSFORMATION are named:

- MY_LS_TRANSFORMATION
- MY_LS_TRANSFORMATION_1

Any indexes that you create on the template tables must conform to the following naming convention:

template_table_name | | suffix

The Transformation ID interface restricts transformation output table names to 26 characters. The Transformation engine appends one of the following suffixes to Risk Manager Transformation output tables:

- _C
- $C
- _G
- $G

Because of the suffix, Risk Manager Transformation output table names are limited to 28 characters or less. Therefore, in order to ensure uniqueness of all secondary template indexes created on an output table, observe the following guidelines:

- The suffix for indexes created on the Risk Manager template tables is limited to two characters.
- The suffix for indexes created on the Ledger_Stat template tables is limited to four characters.

If you create template indexes with a longer suffix some of the secondary template indexes may fail to be created at the end of the transformation process due to naming conflicts. However, only failure to create the primary index causes the transformation to fail.

The OTHER_LEAF_COLUMNS Placeholder Column

The OTHER_LEAF_COLUMNS column in each template is a placeholder column. In the creation of the output tables, this column is replaced by one or more leaf columns, depending on the transformation type:

- For Ledger_Stat transformations, it is replaced with all of the other leaves that are defined in OFSA_CATALOG_OF_LEAVES for inclusion in the LEDGER_ STAT table.
- For Risk Manager Result Detail transformations, it is replaced by the product leaf column used in generating the source Result Detail table.

You can include the OTHER_LEAF_COLUMNS column in the additional template indexes that you create on any of the template tables. When the index is created, this place-holder column is replaced by the column or set of columns whose place is held by it in the transformation template table. This works for a single-column index on the OTHER_LEAF_COLUMNS column, as well as on compound indexes that include that column.

User-Defined Indexes

For some Transformation ID output tables, you may want to create additional indexes, beyond those that are defined by the template indexes, that apply to all output tables of that type.

In the event that processing for the existing output table requires that the table be recreated, the Transformation ID attempts to recreate all user-defined indexes on the output table.

Indexes and Dimension Filters

The Transformation ID allows one or more of the leaf columns to be excluded from the transformation. This is known as a dimension filter. The output table for a Transformation ID that includes a dimension filter is created without the filtered leaf columns. For any template index or user-defined index that contains one or more leaf columns that have been excluded by the dimension filter, the Transformation ID attempts to create the index on the output table without the filtered columns. If all of the columns in an index are excluded by the dimension filter, then that index is not created or recreated for the output table.

A Single Index for the Tree Rollup Transformation

The Tree Rollup transformation is different from the other three transformations. It does not use template indexes. It creates a single, unique index on the column in the output table that replaces the LEAF_COLUMN column in the OFSA_TRANSFORM_ROLLUP_TEMPLATE table.

For example, in the output table for a transformation of a Tree Rollup that is defined on the Org Unit leaf, the ORG_UNIT_ID column replaces the LEAF_COLUMN column. A single, unique index is created on the ORG_UNIT_ID column in the output table.

Table and Index Physical Storage Defaults

There are two tables in the OFSA database that enable you to control the physical storage parameters for output tables and their indexes. These are:

- OFSA_TABLE_STORAGE_DEFAULTS
- OFSA_INDEX_STORAGE_DEFAULTS

These tables enable you to specify all of the physical storage parameters allowed by the CREATE TABLE and CREATE INDEX statements, respectively. The exception to this is partitioned tables. The Transformation ID does not currently support transformations into partitioned output tables. Additionally, for the Ledger Stat transformation, these tables enable you to control how the INITIAL and NEXT parameters are computed during the transformation process.

Because these two tables are similar, the material in the following sections refers to both tables unless otherwise noted, and references to storage parameters for output tables apply equally to indexes.

The Storage Defaults Tables

The following table lists and describes the column names for the OFSA_TABLE_STORAGE_DEFAULTS and OFSA_INDEX_STORAGE_DEFAULTS tables. The column names and descriptions are identified for both tables with the following exception: The OFSA_INDEX_STORAGE_DEFAULTS table does not contain the PCT_USED nor the CACHE columns.

Column Name	Null?	Type
OUTPUT_TABLE_CLASS_CD	NOT NULL	NUMBER (5)

Column Name	Null?	Type
USER_NAME		VARCHAR2 (30)
TABLE_SPACE_NAME		VARCHAR2 (30)
PCT_FREE		NUMBER (2)
PCT_USED		NUMBER (2)
INI_TRANS		NUMBER (5)
MAX_TRANS		NUMBER (5)
MIN_EXTENT_SIZE		NUMBER (14)
MAX_EXTENT_SIZE		NUMBER (14)
DESIRED_EXTENTS		NUMBER (5)
INITIAL_EXTENT		NUMBER (14)
NEXT_EXTENT		NUMBER (14)
MIN_EXTENTS		NUMBER (5)
MAX_EXTENTS		NUMBER (5)
PCT_INCREASE		NUMBER (5)
FREELISTS		NUMBER (5)
FREELIST_GROUPS		NUMBER (5)
LOGGING		VARCHAR2 (3)
DEGREE		VARCHAR2 (10)
INSTANCES		VARCHAR2 (10)
CACHE		VARCHAR2 (5)
BUFFER_POOL		VARCHAR2 (7)

The unique key for the storage defaults tables is OUTPUT_TABLE_CLASS_CD + USER_NAME. However, USER_NAME can be null. This enables you to define one row for each OUTPUT_TABLE_CLASS_CD value with a null USER_NAME, and many rows for each OUTPUT_TABLE_CLASS_CD with the USER_NAME filled in.

In the following sections, type-level row refers to any row with a null USER_NAME; user-level row refers to any row having a value for USER_NAME.

The following sections explain the usage for each of these columns.

Storage Defaults by Transformation Type

You can enter one type-level row per transformation output table type in each of the storage default tables. The transformation output table type is identified by the OUTPUT_TABLE_CLASS_CD value in the FDM Metadata for output tables of that type.

Type-level rows are keyed by OUTPUT_TABLE_CLASS_CD only, with a null value in the USER_NAME column.

Transformation Output Table Type	Output Table Class Cd
Transformed Ledger_Stat	220
Transformed RM Cash Flow	230
Transformed RM Cash Flow Consolidated	430
Transformed RM GAP	240
Transformed RM GAP Consolidated	440
Transformed Tree Rollup	250

The FDM database upgrade initializes one row in each of the storage defaults tables for each transformation output table type with default values. You should review the default values and customize them to your specific situation.

Storage Defaults by Transformation Type + User

You can create additional user-level rows in the storage defaults tables, defined by USER_NAME within each transformation type. This feature provides finer control over the use of the default storage parameters and is particularly helpful for tablespace management.

For example, you have 10 users and each is creating a few small-to-medium-sized output tables of the Ledger_Stat transformation type. You also have one user (Scott) who is creating multiple large-sized tables of that type. The first 10 users might share a single tablespace for their output tables, while Scott's output tables could be assigned to a separate tablespace.

Ordering the Parameter Definition Levels

The ordering of the default storage parameters is from most specific to most general, individually by parameter. That is, any parameter defined at the user level is first; if not defined there, then the parameter defined at the type level is used; if it is not defined at the type level, then the parameter is either computed or defaulted by the Transformation ID process.

Referring to the example, the type-level row defines the parameter values for the output tables for the first ten users and the user-level row, containing SCOTT in the USER_NAME column, defines the parameters values for Scott's output tables. Not all of the parameters need to be specified for Scott. You may want to direct his output tables (or indexes) to a different tablespace and leave all of the other columns null. For those parameters, the values from the type-level tables are used when creating Scott's output tables or indexes.

Physical Storage for the Ledger Stat Transformation

The Risk Manager and Tree Rollup transformations have relatively small freespace requirements. Default values for the INITIAL and NEXT parameters for these transformations are initialized by the FDM database upgrade in the storage defaults tables as described in the section entitled "Table and Index Physical Storage Defaults" in this chapter. You should monitor the average space requirements for these transformation types and adjust these values accordingly.

The Ledger Stat transformation type typically requires much larger amounts of freespace. In order to help you size these tables correctly, the transformation engine can estimate the amount of freespace required for each output table that it creates for a Ledger Stat transformation.

Fitting Into Available Freespace

For the Ledger Stat transformation, the Transformation ID engine estimates the minimum amount of freespace required in the designated tablespace to create each table or index. If it is unable to pre-allocate all of the estimated minimum required freespace, either due to insufficient freespace or freespace fragmentation, then the object is not created.

Failure to create the output table, or failure to create the unique template index (if one is defined) constitutes a fatal error and the transformation is not performed. Failure to create secondary template indexes or user-defined indexes does not result in an error and the transformation proceeds.

If the freespace is fragmented such that the entire space required cannot be pre-allocated using the current values for the INITIAL and NEXT storage parameters (whether specified in the applicable storage defaults table or whether they are computed by the Transformation ID), then the extent sizes are reduced, the number of extents increased and the freespace allocation is attempted again. This process continues until it is determined that the estimated minimum required freespace can be allocated in the available freespace, or until one of the following error conditions is reached:

- INITIAL drops below the value specified for MIN_EXTENT_SIZE
- The number of extents exceeds the value specified for MAX_EXTENTS.

All of the estimated required space may not actually be pre-allocated at the time the table is created. Only the number of extents specified by the MINEXTENTS parameter are actually allocated when the table is created.

Computing INITIAL and NEXT Storage Parameters

For the Ledger Stat transformation type, the determination of the INITIAL and NEXT storage parameters, used to create an output table or one of its indexes, are described in the following table. MIN_EXTENT_SIZE, MAX_EXTENT_SIZE, INITIAL_EXTENT, and NEXT_EXTENT are expressed in bytes, as in the USER_TABLES and USER_INDEXES catalog views.

Storage Parameter	Parameter Conditions
INITIAL_EXTENT	If INITIAL_EXTENT is greater than 0, the Transformation ID attempts to use this value *as-is* for the INITIAL parameter.
	If INITAL_EXTENT equals 0 or is null, the Transformation ID computes the value for the INITIAL parameter, based on the value of DESIRED_EXTENTS. If DESIRED_EXTENTS is not specified, then the Transformation ID attempts to allocate all of the required freespace in a single initial extent.
DESIRED_EXTENTS	If DESIRED_EXTENTS is greater than 1, then this storage parameter controls the computation of the INITIAL and NEXT storage parameters. The Transformation ID attempts to create DESIRED_EXTENTS number of extents of equal size, totaling the required freespace. INITIAL and NEXT are both set to the freespace required, divided by DESIRED_EXTENTS.

Storage Parameter	Parameter Conditions
NEXT_EXTENT	If INITIAL is computed using DESIRED_EXTENTS, then any value specified for NEXT_EXTENT is overridden and the NEXT parameter is set equal to the computed value for the INITIAL parameter. If NEXT_EXTENT is greater than 0, and DESIRED_EXTENTS is less than or equal to 1 or is not specified, the Transformation ID attempts to used the specified NEXT_EXTENT value *as-is* for the NEXT parameter. If NEXT_EXTENT is 0, and DESIRED_EXTENTS is less than or equal to 1 or is not specified, the NEXT parameter is computed as a predetermined percentage (currently 10%) of the specified or computed value for the INITIAL parameter.
MIN_EXTENT_SIZE	This value is used only if the INITIAL parameter is computed. INITIAL is set to MIN_EXTENT_SIZE if its computed value is less than MIN_EXTENT_SIZE.
MAX_EXTENT_SIZE	This value is used only if the INITIAL parameter is computed. INITIAL is set to MAX_EXTENT_SIZE if its computed value is greater than MAX_EXTENT_SIZE.
MIN_EXTENTS	Determines the number of extents that are pre-allocated for the table when it is created. Computed or supplied values for INITIAL and NEXT are rounded up to the next multiple of five Oracle data blocks.

Usage Summary

Depending on whether or not you specify values for INITIAL_EXTENT and DESIRED_EXTENTS in an applicable row in the applicable storage defaults table, the Transformation ID determines the sizes of the extents for a Ledger Stat transformation output table or index in one of three possible ways:

Method 1

If INITIAL_EXTENT is specified, the Transformation ID uses that value for the INITIAL parameter when creating the table or index.

Method 2

If neither INITIAL_EXTENT nor DESIRED_EXTENTS is specified, the Transformation ID computes the INITIAL parameter, based on the estimated

amount of freespace required, and attempts to create a single initial extent containing all of the required freespace.

Method 3

If INITIAL_EXTENT is not specified and DESIRED_EXTENTS is specified, the Transformation ID attempts to create DESIRED_EXTENTS number of equally-sized extents sufficient to meet or exceed the freespace requirement.

These three methods are explained in detail, as follows:

Method 1 Detailed: INITIAL_EXTENT is specified (value greater than 0).

In this case, the INITIAL parameter is taken from the INITIAL_EXTENT value. The NEXT parameter is taken from the NEXT_EXTENT value if it is specified, otherwise it is computed as a percentage (10%) of the INITIAL parameter.

The INITIAL and NEXT parameter values are then rounded up to the nearest multiple of 5 Oracle data blocks. Only the number of extents specified by the MIN_EXTENTS parameter are pre-allocated when the table is created.

Method 2 Detailed: INITIAL_EXTENT is not specified (value is null or 0) and DESIRED_EXTENTS is not specified (value is null or is less than or equal to 1).

In this case, the INITIAL parameter is set equal to the estimated freespace required. If this is less than MIN_EXTENT_SIZE or greater than MAX_EXTENT_SIZE, then INITIAL is adjusted to be in this range. If the NEXT_EXTENT column is specified, then it is used *as-is* for the NEXT parameter, otherwise the NEXT parameter is computed as a percentage (10%) of the computed INITIAL value.

The INITIAL and NEXT parameter values are then rounded up to the nearest multiple of 5 Oracle data blocks. Only the number of extents specified by the MIN_EXTENTS parameter are pre-allocated when the table is created.

Method 3 Detailed: INITIAL_EXTENT is not specified (value is null or 0) and DESIRED_EXTENTS is specified (value is greater than 1).

In this case, the INITIAL parameter is set equal to the estimated freespace required, divided by DESIRED_EXTENTS. If the result is less than MIN_EXTENT_SIZE or greater than MAX_EXTENT_SIZE, then INITIAL is adjusted to be in this range. The NEXT parameter is then set equal to the INITIAL parameter.

The INITIAL and NEXT parameter values are then rounded up to the nearest multiple of 5 Oracle data blocks. Only the number of extents specified by the MIN_EXTENTS parameter are pre-allocated when the table is created.

As explained in "Fitting Into Available Freespace", the Transformation ID attempts to allocate the required freespace using the parameter settings resulting from one of the three cases described. If the total freespace in the tablespace is greater than the space required for the transformation, but cannot be allocated using the current parameter settings due to freespace fragmentation, then the current number of extents is doubled and treated as a value for DESIRED_EXTENTS. In this scenario the INITIAL and NEXT values are recomputed, according to method 3.

This process is continued until the freespace allocation is successful, or until one of the following error conditions is reached:

- The value of INITIAL is less than MIN_EXTENT_SIZE
- The number of extents exceeds the MAX_EXTENTS storage default parameter

Recommended Usage

For Ledger Stat transformations, it is recommended that you leave the INITIAL and NEXT default storage parameters null or 0, so that they are estimated by the Transformation ID. If your end users are transforming large datasets, then you should use the DESIRED_EXTENTS column to direct the Transformation ID to break the storage allocation into several equally sized extents, rather than allocating all of the required freespace into one large initial extent.

It is good for the transformation engine to pre-allocate the majority of the required freespace for an output table or index so that the space is guaranteed to be there when it is needed. However, in order to avoid wasted storage space, you should set the value of the MIN_EXTENTS parameter to about 90% of the DESIRED_EXTENTS parameter.

For any transformation type, if you find that storage space has been over-allocated, you can use the ALTER TABLE or ALTER INDEX command to re-allocate the unused space and return it to the freespace pool.

The following storage defaults columns correspond to the CREATE TABLE or CREATE INDEX parameters:

- TABLESPACE_NAME
- PCT_FREE
- PCT_USED (OFSA_TABLE_STORAGE_DEFAULTS only)

- INI_TRANS
- MAX_TRANS
- MAX_EXTENTS
- PCT_INCREASE
- FREELISTS
- FREELIST_GROUPS
- LOGGING
- DEGREE
- INSTANCES
- CACHE (OFSA_TABLE_STORAGE_DEFAULTS only)
- BUFFER_POOL

The values for these columns are the same as those that can be seen when querying the USER_TABLES and USER_INDEXES catalog views. They are used directly in the CREATE TABLE and CREATE INDEX SQL commands issued by the Transformation ID process. Refer to the *Oracle8i SQL Reference* for further information on these parameters.

It is recommended that you examine the default values for these columns posted by the FDM Database Upgrade for each of the transformation output table types, and adjust them to suit your needs. You may find the Data Verification ID helpful for editing the values in these tables and for creating new entries for individual users.

Creating Transformation Output Tables and Indexes: An Example

The following example illustrates the concepts explained in the previous sections regarding the creation of transformation output tables and indexes.

For this example, assume that your LEDGER_STAT table has the leaf columns TP_PROD_ID and RM_COA_ID. However, the Transformation ID that you run for this example contains a dimension filter that excludes the RM_COA_ID leaf column from the transformation. Assume, also, that the Data Filter for the Transformation ID selects the Financial Elements 60, 100, 130, 140. The Column Name values for these Financial Elements in Leaf Setup are BEG_BAL, END_BAL, END_T_RATE and AVG_BAL respectively.

You have created two additional template indexes on the OFSA_TRANSFORM_LS_TEMPLATE table. Including the unique index created on the template table by the OFSA database upgrade, the indexes on OFSA_TRANSFORM_LS_TEMPLATE are:

INDEX_NAME	UNIQUE?	COLUMN_NAME
OFSA_TRANSFORM_LS_TEMPLATE	YES	END_DATE
		ORG_UNIT_ID
		GL_ACCOUNT_ID
		COMMON_COA_ID
		OTHER_LEAF_COLUMNS
		CONSOLIDATION_CD
OFSA_TRANSFORM_LS_TEMPLATE_1	NO	ORG_UNIT_ID
OFSA_TRANSFORM_LS_TEMPLATE_2	NO	OTHER_LEAF_COLUMNS

The Transformation ID to be run creates an output table called OFSA_LEDGER_STAT_1. This transformation has been run before, so the output table already exists. After its initial creation, create an index called OFSA_LEDGER_STAT_GL on the GL_ACCOUNT_ID column of OFSA_LEDGER_STAT_1. The definition of this Transformation ID indicates that the output table should be dropped and recreated, and you want the OFSA_LEDGER_STAT_GL index on the existing output table to be preserved.

For all users besides Scott who run transformations that create output tables of the Ledger_Stat output type, you have directed those output tables to be created in the LEDGER_STAT_TS tablespace. Scott is running most of the transformations, so you have given him his own tablespace, called SCOTT_TS.

You have decided to allow the Transformation ID to compute the INITIAL and NEXT extent parameters based on the estimated space requirement for the transformed output table, so you have left the INITIAL_EXTENT and NEXT_EXTENT columns null.

You know that most of the output tables created by users other than Scott is relatively small, so you have left DESIRED_EXTENTS = 1 so that the Transformation ID attempts to allocate all of the required freespace into the initial extent.

Some of the Ledger_Stat output tables that Scott creates are quite large, so you have entered the value of 20 for the DESIRED_EXTENTS parameter in the user-level row, so that the Transformation ID pre-allocates 20 equally-sized extents for each of Scott's Ledger_Stat output tables. You decide to pre-allocate 90% of the estimated required freespace for Scott's tables, so you set MIN_EXTENTS = 18 in Scott's user level row.

The relevant rows in the OFSA_TABLE_STORAGE_DEFAULTS table are listed. The storage default rows for the Ledger_Stat transformation type are identified by OUTPUT_TABLE_CLASS_CD = 220.

Column Name	Values in the Type-level Row	Values in the User-level Row
OUTPUT_TABLE_ CLASS_CD	220	220
USER_NAME		SCOTT
TABLESPACE_NAME	LEDGER_STAT_TS	SCOTT_TS
PCT_FREE	10	
PCT_USED	40	
INI_TRANS	2	
MAX_TRANS	255	
MIN_EXTENT_SIZE	1048576	
MAX_EXTENT_SIZE	1073741824	
DESIRED_EXTENTS	1	20
INITIAL_EXTENT		
NEXT_EXTENT		
MIN_EXTENTS	1	18
MAX_EXTENTS	505	
PCT_INCREASE	0	
FREELISTS	1	
FREELIST_GROUPS	1	
LOGGING	YES	
DEGREE	1	

Column Name	Values in the Type-level Row	Values in the User-level Row
INSTANCES	1	
CACHE	N	
BUFFER_POOL	DEFAULT	

Assume that the required space for the transformation, as estimated by the Transformation ID, is 24,125,440 bytes and that the freespace in the appropriate tablespace is sufficient. In the following examples, the TP_PROD_ID (in bold) has replaced the OTHER_LEAF_COLUMNS column from the template table. Also, the RM_COA_ID column is not in the output table because it has been filtered by the dimension filter. You also see differences in the INITIAL, NEXT, MINEXTENTS and TABLESPACE parameters (all in bold) between the table created by Scott and the table created by the other users.

The Transformation ID generates the following CREATE TABLE statement to create the Ledger_Stat output table when a user other than Scott runs the ID:

```
CREATE TABLE OFS_LEDGER_STAT_1 (
END_DATE          DATE NOT NULL
ORG_UNIT_ID       NUMBER(14) NOT NULL,
GL_ACCOUNT_ID     NUMBER(14) NOT NULL,
COMMON_COA_ID     NUMBER(14) NOT NULL,
TP_PROD_ID        NUMBER(14) NOT NULL,
CONSOLIDATION_CD  NUMBER(5) NOT NULL,
BEG_BAL           NUMBER (16,4) DEFAULT 0,
END_BAL           NUMBER (16,4) DEFAULT 0,
END_T_RATE        NUMBER (16,4) DEFAULT 0,
AVG_BAL           NUMBER (16,4) DEFAULT 0)
PCTFREE 10 PCTUSED 40 INITRANS 2 MAXTRANS 255
STORAGE (INITIAL 24125440 NEXT 2416640
         MINEXTENTS 1 MAXEXTENTS 505
         PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
         BUFFER_POOL DEFAULT)
LOGGING
TABLESPACE LEDGER_STAT_TS
NOPARALLEL
NOCACHE
```

If Scott runs the same ID, the following CREATE TABLE statement is generated:

```
CREATE TABLE OFS_LEDGER_STAT_1 (
END_DATE          DATE NOT NULL,
ORG_UNIT_ID       NUMBER(14) NOT NULL,
GL_ACCOUNT_ID     NUMBER(14) NOT NULL,
COMMON_COA_ID     NUMBER(14) NOT NULL,
TP_PROD_ID        NUMBER(14) NOT NULL,
CONSOLIDATION_CD  NUMBER(5)  NOT NULL,
BEG_BAL           NUMBER (16,4) DEFAULT 0,
END_BAL           NUMBER (16,4) DEFAULT 0,
END_T_RATE        NUMBER (16,4) DEFAULT 0,
AVG_BAL           NUMBER (16,4) DEFAULT 0)
PCTFREE 10 PCTUSED 40 INITRANS 2 MAXTRANS 255
STORAGE (INITIAL 1228800 NEXT 1228800
         MINEXTENTS 18 MAXEXTENTS 505
         PCTINCREASE 0
         FREELISTS 1 FREELIST GROUPS 1
         BUFFER_POOL DEFAULT)
LOGGING
TABLESPACE SCOTT_TS
NOPARALLEL
NOCACHE
```

Regardless of who runs the ID, the Transformation ID creates the following indexes on the output table at the end of the ID processing:

```
INDEX_NAME             UNIQUE? COLUMN_NAME
---------------------- ------- -----------------
OFS_LEDGER_STAT_1      YES     END_DATE
                               ORG_UNIT_ID
                               GL_ACCOUNT_ID
                               COMMON_COA_ID
                               TRUE_GL_ID
                               CONSOLIDATION_CD
OFS_LEDGER_STAT_1_1    NO      ORG_UNIT_ID

OFS_LEDGER_STAT_1_2    NO      TP_COA_ID

OFS_LEDGER_STAT_1_GL   NO      GL_ACCOUNT_ID
```

OFS_LEDGER_STAT_1, OFS_LEDGER_STAT_1_1, and OFS_LEDGER_STAT_1_2 are created from the template index definitions. OFS_LEDGER_STAT_1_GL is created from the SYSTEM_INDEX_INFO definition for that user-defined index, which existed on the output table prior to its recreation by the Transformation ID.

The template index on the OTHER_LEAF_COLUMNS column (TRANSFORM_LS_TEMPLATE_2) translates into the OFS_LEDGER_STAT_1_2 index on TP_COA_ID. RM_COA_ID is excluded by the dimension filter, so it is not included in the index.

Routine Cleanup

Dropping Obsolete Transformation Output Tables

As the FDM Administrator, you are responsible for managing the freespace in the tablespaces. Periodically unregister and drop old transformation output tables that are no longer needed by your users. Use the FDM Administration application to unregister such tables from the FDM Metadata. Once this is done, you can drop the table (either directly in SQL*Plus or through Enterprise Manager). Dropping the table releases the space previously occupied by the table so that it can be reused for other objects.

Deleting from OFSA_STP

All DDL and DCL operations, as well as some DML operations performed during Transformation ID processing are logged in the OFSA_STP table. It is recommended that you include a step at the beginning of your monthly processing cycle to delete older rows from this table.

Transformation ID Error Recovery

When a Transformation ID begins processing, it reserves exclusive use of the output table until processing is complete, at which time it unreserves the table. This means that only one Transformation ID can process against the output table at one time. Output table reservation is accomplished by posting the name of the output table, the process start time, the TRANSFORM_SYS_ID value of the Transformation ID reserving the table and the USER_NAME of the current user to a row in the OFSA_TABLE_TRACKING table.

Unreservation of the output table upon completion of the transformation is accomplished by posting the process end time to that same row and filling in other columns in the row that indicate the source of the transformed data and the type of operations performed by the ID processing.

If for some reason the output table cannot be created, unreservation of the table is accomplished during ID processing by deleting the appropriate row from the OFSA_TABLE_TRACKING table.

In the event of a hardware failure or power outage while the Transformation ID is processing, an output table can remain reserved, even though processing has not been completed. The table remains reserved to the original Transformation ID and user when a failure occurs.

You have two options to complete the process and unreserve the output table. The first option is to rerun the same ID from the same user. Other users and other Transformation IDs are locked out. The second option is to delete the appropriate row from the OFSA_TABLE_TRACKING table. This unreserves the output table and allows it to be processed by another user and/or a different Transformation ID.

Temporary Objects

Some of the OFSA applications create temporary database objects, including tables and views. Sometimes these temporary objects are not properly deleted when the process that created them is complete. For example, this can happen when the user's session is interrupted by a network failure. FDM uses a tracking table called OFSA_TEMP_OBJECTS to help you identify temporary objects that are no longer needed and can be deleted. Every time an OFS application creates a temporary object, the creation of the object is recorded in the tracking table. When the application is finished with the temporary object, the record of the object is dropped from the tracking table. Objects listed in the tracking table that are not being used by any process can be safely deleted.

Message and Audit Objects

FDM provides tables for storing error and audit messages for FDM and OFS processing operations. Because data in the Message Objects increases when processing operations are performed in FDM, it is important to monitor the number and size of extents for these tables.

Note: Over time, the size of the Audit and Message tables tend to grow. It is recommended that you include a step at the beginning of your monthly processing cycle to delete older rows from these tables.

Audit Tables

Audit tables store detail information for debugging OFS processing operations.

OFSA_AUDIT_TRAIL

Performance Analyzer Allocation processes write output data to the OFSA_AUDIT_TRAIL table when the Audit Trail checkbox in marked in the Allocation ID. This table stores information about the data records updated and inserted by the Allocation process.

The AS_OF_DATE column in this table designates the process date for which the audit information applies. Use the AS_OF_DATE column to identify old and obsolete audit information.

OFSA_PROCESS_CASH_FLOWS

The OFSA_PROCESS_CASH_FLOWS table is used whenever the write cash flows option is turned on when performing cash flow calculations. All records in this table are identified by the process that creates them. This information is in the RESULT_SYS_ID column. You may have situations in which the process that created the records in this table no longer exists. In these cases, the obsolete records should be removed.

To perform this operation, run the following SQL statement on the database:

```
Delete from OFSA_PROCESS_CASH_FLOWS
where RESULT_SYS_ID not in
(select  SYS_ID_NUM from OFSA_CATALOG_OF_IDS where ID_TYPE in (204,205));
```

OFSA_STP

OFS applications call stored procedures to create or drop temporary objects. These same procedures manage entries in OFSA_TEMP_OBJECTS. The stored procedures have the necessary permissions to create and drop temporary objects and therefore individual users do not require these permissions.

The OFSA_STP table logs DDL and DCL operations performed by these stored procedures. Operations such as creating RM Result Detail tables or Transformation output tables, granting table or role privileges, creating public synonyms and creating and dropping temporary objects are included in this table.

The Utility procedures (described in Chapter 21, "FDM Utilities") log audit entries in OFSA_STP. The FDM Administration Grant procedures also log grant statements into this table.

The OFSA_STP table is available for your use as well as Oracle Support Services to resolve problems related to these operations.

Message Tables

Message tables provide warning and error messages for OFS processing operations.

OFSA_PROCESS_ERRORS

The OFSA_PROCESS_ERRORS table stores informational, error and warning messages for Risk Manager, Transfer Pricing, and Data Correction Processing.

OFSA_MESSAGE_LOG

This table stores informational, error, and warning messages for Performance Analyzer and Rate Manager processing. Market Manager also creates entries in this table.

Packages, Procedures, and Java Classes

The FDM database includes PL/SQL packages and procedures, as well as Java Classes. All of these are loaded into the database by either the FDM database creation process or database upgrade process. These objects are a required component of the FDM database definition.

It is possible for one or more of these objects to become INVALID. This can occur during a database import, or because an object reference by the package, procedure or java class no longer exists in the database. You or your users may receive Oracle errors indicating this situation with such errors as:

- ORA-04068 Existing state of packages has been discarded
- Unable to resolve Java Class

To identify if this is the case, run the following query in SQL*Plus as the FDM Schema Owner:

```
select object_name, object_type, status
from user_objects
where status = 'INVALID';
```

Included in the utilities directory with the FDM database scripts is a script to refresh these objects, in the event that one or more of these objects becomes invalid. The default location for this refresh script is the **OFSA_INSTALL/dbs/<OFSA**

release>/utilities/stp subdirectory of your OFSA installation directory. OFSA_INSTALL is the convention used to indicate where the OFSA software is installed in your directory structure.

Financial Data Manager Packages

The Financial Data Manager packages consist of all packages, procedures and java classes required by the OFS applications, excluding those required exclusively by Market Manager.

To run the script, go to this directory location and login to SQL*Plus as the FDM Schema Owner. Then type the following:

```
<SQL> @fdm_packages.sql
```

The fdm_packages.sql script prompts for the password of the FDM Schema Owner. This is necessary for creating the Java Classes in the database.

This script creates the following log files in the **OFSA_INSTALL/dbs/<OFSA release>/log** directory:

- fdm_packages.log
- pass_jar_status.log
- rtm_jar.log

Review these logs for any errors.

Market Manager Packages

The Market Manager packages consist of packages and stored procedures required by Market Manager, including some that are also required by FDM Administration.

To run the script, go to this directory location and login to SQL*Plus as the FDM Schema Owner. Then type the following:

```
<SQL> @mm_packages.sql
```

This script creates the following log files in the **OFSA_INSTALL/dbs/<OFSA release>/log** directory:

- mm_packages.log

Review this log for any errors.

Views and Triggers

The FDM database includes both views and triggers. These objects are loaded into the database by either the FDM database creation process or database upgrade process. These objects are a required component of the FDM database definition.

It is possible for one or more of these objects to become INVALID. This can occur during a database import, or because an object reference by the package, procedure or java class no longer exists in the database. You or your users can receive Oracle errors indicating this situation with such errors as:

- ORA-04063: view has errors
- Errors or incorrect behavior during insert, update or deleting on view objects

To identify if this is the case, run the following query in SQL*Plus as the FDM Schema Owner:

```
select object_name, object_type, status
from user_objects
where status = 'INVALID';
```

Included in the utilities directory with the FDM database scripts is a script to refresh these objects, in the event that one or more of these objects becomes invalid. The default location for this refresh script is the **OFSA_INSTALL/dbs/<OFSA release>/utilities/views** subdirectory of your OFSA installation directory. OFSA_INSTALL is the convention used to indicate where the OFSA software is installed in your directory structure.

Financial Data Manager Views

The Financial Data Manager views consist of all views required by the OFS applications, excluding those required exclusively by Market Manager.

To run the script, go to this directory location and login to SQL*Plus as the FDM Schema Owner. Then type the following:

```
<SQL> spool fdm_views.log
<SQL> @fdm_views.sql
```

Review the log file for any errors. Because the fdm_views.sql script drops and re-creates views, the privileges for these objects need to be regranted. After the script completes successfully, run the FDM Grant All procedure to regrant privileges for these objects.

Market Manager Views

The Market Manager views consists of views required by Market Manager.

To run the script, go to this directory location and login to SQL*Plus as the FDM Schema Owner. Then type the following:

```
<SQL> spool mm_views.log
<SQL> @mm_views.sql
```

Review the log file for any errors. After the script completes successfully, run the Grant All procedure. Because the mm_views.sql script drops and re-creates views, the privileges for these objects need to be regranted.

Seeded Data Tables and Ranges

Oracle defines seeded data as data:

- Placed in the database to run the OFS applications properly
- Composing the IDs shipped with the database

Seeded tables are designated as either:

- FDM Metadata
- Seeded Range Reserved
- Seeded Unreserved

Note: All tables seeded by FDM are Reserved tables. The designation of Seeded Unreserved indicates that users are allowed to edit and modify the data that FDM seeds into these tables.

Data in Seeded Range Reserved tables is deleted and re-loaded every upgrade. Any user data within the designated reserved range is therefore deleted by the upgrade process. Data in tables designated as Seeded Unreserved is populated only once by the database creation process. Users are allowed to enter their own data within the Seeded Unreserved tables without restrictions. FDM Metadata tables are populated by the database creation process, but are updated by the database upgrade process for any rows pertaining to FDM Reserved Objects.

FDM Metadata Seeded Tables

FDM Metadata tables are seeded by the database creation process and are updated by the database upgrade process for any rows pertaining to FDM Reserved Objects. Rows for FDM User Defined Objects are not modified by the database upgrade process.

OFSA_COLUMN_PROPERTIES	OFSA_ROLE_ASSIGNMENT
OFSA_COLUMN_REQUIREMENTS	OFSA_ROLES
OFSA_COLUMN_REQUIREMENTS_MLS	OFSA_TABLES
OFSA_DB_OBJECT_PRIV_ASSIGNMENT	OFSA_TABLES_MLS
OFSA_DB_SYS_PRIV_ASSIGNMENT	OFSA_TAB_COLUMNS
OFSA_DESCRIPTION_TABLES	OFSA_TAB_COLUMNS_MLS
OFSA_PRIVILEGE_RECIPIENTS	OFSA_TABLE_CLASS_ASSIGNMENT

Seeded Range Reserved

The seeded data in the tables and ranges described in this section are deleted and reloaded with the master seeded data during every upgrade within the designated reserved data range.

Caution: Data inserted into these tables within the designated ranges by your users is deleted when a database upgrade is run.

Seeded Range Reserved tables consist of:

- FDM and Market Manager Shared Tables
- FDM Only Tables
- Market Manager Only Tables

Range Reserved FDM and Market Manager Shared Tables

Shared FDM and Market Manager tables are present in both Market Manager and Financial Data Manager installations.

Table Name	Reserved Range
AGGREGATION	Entire Table
HARV_MSG_CLASS	Entire Table
HARV_MSG_TEXT	Entire Table
HARV_OBJECT_PRIVS	Entire Table
HARV_ROLE	Entire Table
HARV_SYSTEM_PRIVS	Entire Table
OFSA_JOB_STATUS_CD	Entire Table
OFSA_JOB_STATUS_MLS	Entire Table
OFSA_MESSAGES_B	Entire Table
OFSA_MESSAGES_MLS	Entire Table
OFSA_MSG_SEVERITY_CD	Entire Table
OFSA_MSG_SEVERITY_MLS	Entire Table

Seeded Range Reserved FDM Only Tables

FDM Only Tables are used for Financial Data Manager. These tables are not present in a Market Manager *Standalone* installation.

Table Name	Reserved Range
BANK_INIT	Entire table
OFSA_ACCRUAL_BASIS_CD	Entire table
OFSA_ACCRUAL_BASIS_MLS	Entire table
OFSA_ACCUMULATION_TYPE_CD	Entire table
OFSA_ACCUMULATION_TYPE_MLS	Entire table
OFSA_ACTION_ASSIGNMENT	WHERE protected_flg = 1
OFSA_ACTIONS	Entire table
OFSA_ADJUSTABLE_TYPE_CD	where adjustable_type_cd<500
OFSA_ADJUSTABLE_TYPE_MLS	where adjustable_type_cd<500

Table Name	Reserved Range
OFSA_AMORTIZATION_TYPE_CD	where amortization_type_cd<1000
OFSA_AMORTIZATION_TYPE_MLS	where amortization_type_cd<1000
OFSA_AMOUNT_TYPE_CD	Entire table
OFSA_AMOUNT_TYPE_MLS	Entire table
OFSA_APPLICATIONS	Entire table
OFSA_APPLICATION_CONSTRUCTS	Entire table
OFSA_APP_ASSIGNMENT	WHERE protected_flg = 1
OFSA_APPS_INSTALL_GROUPS	Entire table
OFSA_CALC_SOURCE_CD	Entire table
OFSA_CALC_SOURCE_MLS	Entire table
OFSA_CATALOG_OF_IDS	where sys_id_num>79998 and sys_id_ num<100001
OFSA_COLUMN_PROPERTY_CD	Entire table
OFSA_COLUMN_PROPERTY_MLS	Entire table
OFSA_COL_PROPERTY_REQUIREMENTS	WHERE protected_flg = 1
OFSA_COMPOUND_BASIS_CD	Entire table
OFSA_COMPOUND_BASIS_MLS	Entire table
OFSA_CONSOLIDATION_CD	where consolidation_cd<=350
OFSA_CONSOLIDATION_MLS	where consolidation_cd<=350
OFSA_CONSTRUCTS	Entire table
OFSA_CONSTRUCT_ACTIONS	Entire table
OFSA_CORRECTION_PROC_MSG_CD	where error_code>=9000
OFSA_CORRECTION_PROC_MSG_MLS	where error_code>=9000
OFSA_CURRENCIES	where user_def_flg = 0
OFSA_CURRENCY_MLS	where user_def_flg = 0
OFSA_CURRENCY_STATUS_CD	Entire table
OFSA_CURRENCY_STATUS_MLS	Entire table
OFSA_DATA_TYPE_DSC	Entire table

Table Name	Reserved Range
OFSA_DB_OBJ_PRIVS	Entire table
OFSA_DETAIL_ELEM_B	WHERE leaf_node<10000
OFSA_DETAIL_ELEM_MLS	WHERE leaf_node<10000
OFSA_DETAIL_RECORD_CD	Entire table
OFSA_DETAIL_RECORD_MLS	Entire table
OFSA_DISCOUNT_RATE_METHOD_CD	Entire table
OFSA_DISCOUNT_RATE_METHOD_MLS	Entire table
OFSA_ESTIMATION_SMOOTHING_CD	Entire table
OFSA_ESTIMATION_SMOOTHING_MLS	Entire table
OFSA_EXCHANGE_RATE_STATUS_CD	Entire table
OFSA_EXCHANGE_RATE_STATUS_MLS	Entire table
OFSA_EXCHNG_RATE_CONV_TYPE_CD	Entire table
OFSA_EXCHNG_RATE_CONV_TYPE_MLS	Entire table
OFSA_EXCLUDED_PRIV_RECIPIENTS	Entire table
OFSA_FBAL_BOOKING_CD	Entire table
OFSA_FBAL_BOOKING_MLS	Entire table
OFSA_FBAL_DIMENSION_CD	Entire table
OFSA_FBAL_DIMENSION_MLS	Entire table
OFSA_FBAL_METHOD_CD	Entire table
OFSA_FBAL_METHOD_MLS	Entire table
OFSA_FBAL_RATE_VOLUME_CD	Entire table
OFSA_FBAL_RATE_VOLUME_MLS	Entire table
OFSA_FBAL_RUNOFF_CD	Entire table
OFSA_FBAL_RUNOFF_MLS	Entire table
OFSA_FCAST_IRC_METHOD_CD	Entire table
OFSA_FCAST_IRC_METHOD_MLS	Entire table
OFSA_FCAST_XRATE_METHOD_CD	Entire table
OFSA_FCAST_XRATE_METHOD_MLS	Entire table

Table Name	Reserved Range
OFSA_FINANCIAL_ELEMENTS	Entire table
OFSA_FIN_ELEM_SET	Entire table
OFSA_FIN_ELEM_SET_DTL	Entire table
OFSA_FORWARD_TYPE_CD	Entire table
OFSA_FORWARD_TYPE_MLS	Entire table
OFSA_FREQUENCY_UNIT_CD	Entire table
OFSA_FREQUENCY_UNIT_MLS	Entire table
OFSA_IDT_DATA_FILTER	where sys_id_num>79998 and sys_id_ num<100001
OFSA_IDT_FORMULA	where sys_id_num>79998 and sys_id_ num<100001
OFSA_IDT_REPORT	where sys_id_num>79998 and sys_id_ num<100001
OFSA_IDT_REPORT_COLUMN	where sys_id_num>79998 and sys_id_ num<100001
OFSA_IDT_ROLLUP	where sys_id_num>79998 and sys_id_ num<100001
OFSA_IDT_SQL	where sys_id_num>79998 and sys_id_ num<100001
OFSA_IDT_STRATIFICATION	where sys_id_num>79998 and sys_id_ num<100001
OFSA_IDT_SUBTOTAL	where sys_id_num>79998 and sys_id_ num<100001
OFSA_IDT_USER_DEFINED	where sys_id_num>79998 and sys_id_ num<100001
OFSA_ID_FOLDERS	WHERE protected_flg = 1
OFSA_ID_TYPE_CD	Entire table
OFSA_ID_TYPE_FLAGS	where application>=0
OFSA_ID_TYPE_MLS	Entire table
OFSA_INSTALL_OBJECTS	Entire table
OFSA_INSTRUMENT_TYPE_CD	WHERE instrument_type_cd<500
OFSA_INSTRUMENT_TYPE_MLS	WHERE instrument_type_cd<500

Table Name	Reserved Range
OFSA_INTEREST_TIMING_TYPE_CD	Entire table
OFSA_INTEREST_TIMING_TYPE_MLS	Entire table
OFSA_INT_COMPONENT_TYPE_CD	Entire table
OFSA_INT_COMPONENT_TYPE_MLS	Entire table
OFSA_IRC_FMT_COMPOUND_B_REL	Entire table
OFSA_IRC_FORMAT_CD	Entire table
OFSA_IRC_FORMAT_MLS	Entire table
OFSA_LANGUAGE_MAP	Entire table
OFSA_LEAF_DESC	where leaf_node<10000 and leaf_num_id=0
OFSA_LEVEL_DESC	where sys_id_num>79998 and sys_id_ num<100001
OFSA_LOOKUP	Entire table
OFSA_MLS	Entire table
OFSA_MODIFY_ACTION_CD	Entire table
OFSA_MODIFY_ACTION_MLS	Entire table
OFSA_MULTIPLIER_CD	Entire table
OFSA_MULTIPLIER_MLS	Entire table
OFSA_NET_MARGIN_CD	Entire table
OFSA_NET_MARGIN_MLS	Entire table
OFSA_NODE_DESC	where sys_id_num>79998 and sys_id_ num<100001
OFSA_OPTION_EXERCISE_CD	Entire table
OFSA_OPTION_EXERCISE_MLS	Entire table
OFSA_OPTION_TYPE_CD	Entire table
OFSA_OPTION_TYPE_MLS	Entire table
OFSA_PATTERN_TYPE_CD	Entire table
OFSA_PATTERN_TYPE_MLS	Entire table
OFSA_PAYMENT_TYPE_CD	Entire table
OFSA_PAYMENT_TYPE_MLS	Entire table

Table Name	Reserved Range
OFSA_PMT_PATTERN_TYPE_CD	Entire table
OFSA_PMT_PATTERN_TYPE_MLS	Entire table
OFSA_PP_CALC_METHOD_CD	Entire table
OFSA_PP_CALC_METHOD_MLS	Entire table
OFSA_PP_DIM_TYPE_CD	Entire table
OFSA_PP_DIM_TYPE_MLS	Entire table
OFSA_PP_HYPERCUBE_MAP	Entire table
OFSA_PP_QUOTE_CD	Entire table
OFSA_PP_QUOTE_MLS	Entire table
OFSA_PP_RATE_TERM_CD	Entire table
OFSA_PP_RATE_TERM_MLS	Entire table
OFSA_PROCESSES	Entire table
OFSA_PROCESS_ENGINE	Entire table
OFSA_PROCESS_FILTER_TYPE_CD	Entire table
OFSA_PROCESS_FILTER_TYPE_MLS	Entire table
OFSA_PROCESS_PARTITION_CD	Entire table
OFSA_PROCESS_PARTITION_MLS	Entire table
OFSA_PROPERTY_COLUMNS	WHERE protected_flg = 1
OFSA_PROPERTY_STP	Entire table
OFSA_RATE_CAP_TYPE_CD	Entire table
OFSA_RATE_CAP_TYPE_MLS	Entire table
OFSA_RATE_CHG_ROUNDING_CD	Entire table
OFSA_RATE_CHG_ROUNDING_MLS	Entire table
OFSA_RATE_DATA_SOURCE_CD	Entire table
OFSA_RATE_DATA_SOURCE_MLS	Entire table
OFSA_RATE_FLOOR_TYPE_CD	Entire table
OFSA_RATE_FLOOR_TYPE_MLS	Entire table
OFSA_RECIPIENT_TYPE_DSC	Entire table

Table Name	Reserved Range
OFSA_REG_D_STATUS_CD	Entire table
OFSA_REG_D_STATUS_MLS	Entire table
OFSA_REPORT_LEAVES	where report_leaves_id>79998 and report_ leaves_id<100001
OFSA_REPRICE_METHOD_CD	Entire table
OFSA_REPRICE_METHOD_MLS	Entire table
OFSA_RESULT_TYPE_CD	Entire table
OFSA_RESULT_TYPE_MLS	Entire table
OFSA_ROLL_FACILITY_CD	Entire table
OFSA_ROLL_FACILITY_MLS	Entire table
OFSA_ROW_NUM_ORDER	Entire table
OFSA_SECURITY_PROFILES	Entire table
OFSA_SEC_PROFILE_ASSIGNMENT	Entire table
OFSA_SERVICING_AGENT_CD	Entire table
OFSA_SERVICING_AGENT_MLS	Entire table
OFSA_SETTLEMENT_TYPE_CD	Entire table
OFSA_SETTLEMENT_TYPE_MLS	Entire table
OFSA_SIC_CD	Entire table
OFSA_SIC_MLS	Entire table
OFSA_SMOOTHING_METHOD_CD	Entire table
OFSA_SMOOTHING_METHOD_MLS	Entire table
OFSA_SOLICIT_SOURCE_CD	Entire table
OFSA_SOLICIT_SOURCE_MLS	Entire table
OFSA_STOCH_RANDOM_SEQ_TYPE_CD	Entire table
OFSA_STOCH_RANDOM_SEQ_TYPE_MLS	Entire table
OFSA_STRIKE_TYPE_CD	Entire table
OFSA_STRIKE_TYPE_MLS	Entire table
OFSA_STRINGS_B	Entire table

Table Name	Reserved Range
OFSA_STRINGS_MLS	Entire table
OFSA_SYSTEM_PRIVILEGE_MAP	Entire table
OFSA_TABLE_CLASSIFICATION	Entire table
OFSA_TABLE_CLASSIFICATION_MLS	Entire table
OFSA_TABLE_CLASS_PROPERTIES	Entire table
OFSA_TABLE_PROPERTIES	Entire table
OFSA_TABLE_USAGE	Entire table
OFSA_TERM_TYPE_CD	Entire table
OFSA_TERM_TYPE_MLS	Entire table
OFSA_TM_FORMULA_ELEM	Entire table
OFSA_TM_PROC_TYPE_CD	Entire table
OFSA_TM_PROC_TYPE_MLS	Entire table
OFSA_TP_ASSIGN_DATE_CD	Entire table
OFSA_TP_ASSIGN_DATE_MLS	Entire table
OFSA_TP_CALC_METHOD_CD	Entire table
OFSA_TP_CALC_METHOD_MLS	Entire table
OFSA_TP_CALC_MODE_CD	Entire table
OFSA_TP_CALC_MODE_MLS	Entire table
OFSA_TP_LEAF_DATA_SOURCE_CD	Entire table
OFSA_TP_LEAF_DATA_SOURCE_MLS	Entire table
OFSA_TP_OPT_COST_METHOD_CD	Entire table
OFSA_TP_OPT_COST_METHOD_MLS	Entire table
OFSA_TP_TARGET_BAL_CD	Entire table
OFSA_TP_TARGET_BAL_MLS	Entire table
OFSA_TRANSFORM_PROC_SCOPE_CD	Entire table
OFSA_TRANSFORM_PROC_SCOPE_MLS	Entire table
OFSA_TRANSFORM_SRC_TYPE_CD	Entire table
OFSA_TRANSFORM_SRC_TYPE_MLS	Entire table

Table Name	Reserved Range
OFSA_TS_MODEL_CD	Entire table
OFSA_TS_MODEL_MLS	Entire table
OFSA_USAGE_CD	Entire table
OFSA_USAGE_MLS	Entire table
OFSA_USER_GROUPS	WHERE protected_flg = 1
OFSA_VERSION	Entire table
OFSA_VIRTUAL_TABLES	Entire table
OFSA_VIRTUAL_TABLES_MLS	Entire table
PROF_INIT	Entire table
PSRV_INIT	Entire table
PUPD_RUN	Entire table
PUPD_SERVER	Entire table
PUPD_STEP_MASTER	Entire table
PUPD_TAB	Entire table
PUPD_TAB_COL	Entire table
SALUTATION	Entire table
SRV_INIT	Entire table
STRIDX	Entire table
STRMEM	Entire table
STRUCT	Entire table

Seeded Range Reserved Market Manager Only Tables

Market Manager Only Tables are used only by Market Manager. These tables are not present in an FDM Only installation.

Table Name	Reserved Range
AGGREGATION_DETAILS	Entire Table
AGGR_OPERATOR_DSC	Entire Table

Table Name	Reserved Range
ASSIGNMENT_METHOD_CD	Entire Table
ASSIGNMENT_METHOD_MLS	Entire Table
CAMPAIGN_CALC_SOURCE_CD	Entire Table
CAMPAIGN_CALC_SOURCE_MLS	Entire Table
CAMPAIGN_MEASURE_CD	Entire Table
CAMPAIGN_MEASURE_MLS	Entire Table
OFSA_BATCH_EVENTS_STATUS_CD	Entire Table
OFSA_BATCH_EVENTS_STATUS_MLS	Entire Table
OFSA_BATCH_EVENTS_TYPE_CD	Entire Table
OFSA_BATCH_EVENTS_TYPE_MLS	Entire Table
OFSA_COMPONENT_TYPE_CD	Entire Table
OFSA_COMPONENT_TYPE_MLS	Entire Table
OFSA_FINANCIAL_SCENARIO_CD	Entire Table
OFSA_FINANCIAL_SCENARIO_MLS	Entire Table
QUERY_ROLE_CD	Entire Table
QUERY_ROLE_MLS	Entire Table
QUERY_SOURCE_CD	Entire Table
QUERY_SOURCE_MLS	Entire Table
TRACKING_METHOD_CD	Entire Table
TRACKING_METHOD_MLS	Entire Table
TRACKING_STATUS_CD	Entire Table
TRACKING_STATUS_MLS	Entire Table

Seeded Unreserved

Following is a list of tables seeded by the database creation process that are not protected against update. Users are allowed to insert, update, and delete rows in these tables without restrictions.

Seeded Unreserved tables consist of:

- FDM and Market Manager Shared Tables
- FDM Only Tables
- Market Manager Only Tables

Seeded Unreserved FDM and Market Manager Shared Tables

Shared FDM and Market Manager tables are present in both Market Manager and Financial Data Manager installations.

CYCLE	HARV_TAB
HARV_CONFIG	HARV_TAB_COL
HARV_IND	JOB_PARMS
HARV_IND_COL	

Seeded Unreserved FDM Only Tables

FDM Only Tables are used for Financial Data Manager. These tables are not present in a Market Manager *Standalone* installation.

DEF	OFSA_ISSUER_MLS
DEFMAP	OFSA_LIEN_POSITION_CD
ISEG	OFSA_LIEN_POSITION_MLS
OFSA_CATALOG_OF_LEAVES	OFSA_LIQUIDITY_CLASS_CD
OFSA_CMO_TRANCHE_CD	OFSA_LIQUIDITY_CLASS_MLS
OFSA_CMO_TRANCHE_MLS	OFSA_LOAN_PROPERTY_TYPE_CD
OFSA_COLLATERAL_CD	OFSA_LOAN_PROPERTY_TYPE_MLS
OFSA_COLLATERAL_MLS	OFSA_MARKET_SEGMENT_CD
OFSA_COLLATERAL_STATUS_CD	OFSA_MARKET_SEGMENT_MLS
OFSA_COLLATERAL_STATUS_MLS	OFSA_MORTGAGE_AGENCY_CD
OFSA_COLLATERAL_SUB_TYPE_CD	OFSA_MORTGAGE_AGENCY_MLS
OFSA_COLLATERAL_SUB_TYPE_MLS	OFSA_OCCUPANCY_CD
OFSA_COLLATRL_DISCHRG_TYPE_CD	OFSA_OCCUPANCY_MLS
OFSA_COLLATRL_DISCHRG_TYPE_MLS	OFSA_OVERDRAFT_PROTECTION_CD

OFSA_COMMITMENT_TYPE_CD	OFSA_OVERDRAFT_PROTECTION_MLS
OFSA_COMMITMENT_TYPE_MLS	OFSA_OWNERSHIP_CD
OFSA_CONFORMANCE_CD	OFSA_OWNERSHIP_MLS
OFSA_CONFORMANCE_MLS	OFSA_PLEDGED_STATUS_CD
OFSA_CONTROL	OFSA_PLEDGED_STATUS_MLS
OFSA_CREDIT_RATING_CD	OFSA_PRIVATE_MTG_INSURER_CD
OFSA_CREDIT_RATING_MLS	OFSA_PRIVATE_MTG_INSURER_MLS
OFSA_CREDIT_STATUS_CD	OFSA_PROCESS_DATA_SLICES
OFSA_CREDIT_STATUS_MLS	OFSA_PROCESS_DATA_SLICES_DTL
OFSA_DIRECT_IND_CD	OFSA_PROCESS_ID_RUN_OPTIONS
OFSA_DIRECT_IND_MLS	OFSA_PROCESS_ID_STEP_RUN_OPT
OFSA_DYN_TAB_CLASS_PRIV_ASSIGN	OFSA_PRODUCT_TYPE_CD
OFSA_EXIST_BORROWER_CD	OFSA_PRODUCT_TYPE_MLS
OFSA_EXIST_BORROWER_MLS	OFSA_PURPOSE_CD
OFSA_FISCAL_YEAR_INFO	OFSA_PURPOSE_MLS
OFSA_GEOGRAPHIC_LOC_CD	OFSA_PUT_CALL_CD
OFSA_GEOGRAPHIC_LOC_MLS	OFSA_PUT_CALL_MLS
OFSA_GUARANTOR_RELATION_CD	OFSA_TABLE_EXTENTS
OFSA_GUARANTOR_RELATION_MLS	OFSA_TABLE_STORAGE_DEFAULTS
OFSA_HELD_FOR_SALE_CD	OFSA_UPGRADE_LOG
OFSA_HELD_FOR_SALE_MLS	PUPD_IND
OFSA_INDEX_STORAGE_DEFAULTS	PUPD_IND_COL
OFSA_INTEGRATOR_CONFIG	PUPD_RUN_PREPOST
OFSA_ISSUER_CD	

Seeded Unreserved Market Manager Only Tables

Market Manager Only Tables are used only by Market Manager. These tables are not present in an FDM Only installation.

CONTACT_METDHOD_CD	INCENTIVE_TYPE_MLS
CONTACT_METHOD_MLS	OFSA_MM_SYSTEM_CODE_VALUES
INCENTIVE_TYPE_CD	

17

FDM Leaf Management

In the context of Oracle Financial Services (OFS) applications, a Leaf Column is an column that provides data dimensionality. Leaf Columns enable data categorization via hierarchies (termed *Tree Rollups*) in the OFS applications. They are special columns within Oracle Financial Data Manager (FDM) that are used for a variety of purposes within the OFS applications, including Oracle Performance Analyzer, Oracle Risk Manager, and Oracle Transfer Pricing processing operations. Generally, Leaf Columns are defined for use with Instrument and client data tables, including the LEDGER_STAT table.

This chapter provides information on how to manage Leaf Columns within the FDM database environment. Specific topics in this chapter include:

- Seeded Leaf Columns
- Leaf Registration
- Managing Leaf Values

Seeded Leaf Columns

Included with the FDM database are four Leaf Columns. These columns are:

- Common Chart of Account ID (common_coa_id)
- FInancial Element ID (financial_elem_id)
- General Ledger Account ID (gl_account_id)
- Organizational Unit ID (org_unit_id)

Caution: Do not unregister any of the seeded Leaf Columns. To do so may cause the OFS applications to function improperly.

Leaf Registration

Leaf Registration refers to the identification of new column names as *Leaf Columns*. Once a Leaf Column is registered in the FDM metadata, that column name becomes available for OFS application specific operations, such as processing, or creating business assumption information.

FDM enables you to register any column to be a Leaf Column, as long as it meets certain requirements. These requirements are discussed in this section. If a column does not meet the requirements, the FDM Administration prevents the Leaf Registration operation with an error message.

Note: The concept of Leaf Registration is a separate, but related concept to that of registering a column for an object as FDM Data Type LEAF. In order to register a new Leaf Column, it must first be registered as a column of FDM Data Type LEAF on a designated set of tables.

Note: The maximum number of user-defined Leaf Columns that can be registered on the LEDGER_STAT table is seven. This is in addition to the four FDM seeded Leaf Columns.

Types of Leaf Columns

Leaf Columns are designated as one of the following types:

- Ledger Only
- All (also referred to as Both Instrument and Ledger)

A *Ledger Only* Leaf Column is required to exist only on the LEDGER_STAT table (and a few other application specific internal objects). The seeded Leaf Column Financial Element ID is designated as a Ledger Only leaf. It is unlikely that you will ever need to register any additional columns as Ledger Only.

The more common Leaf Type for user-defined Leaf Columns is that of type All (also referred to as Both Instrument and Ledger). An All Leaf is required to exist on all client data objects that are used for OFS processing operations, such as Allocation, Transfer Pricing, or Risk Manager processing. Generally, this list of objects includes all instrument and account tables registered in the FDM metadata.

Key and Non-key Leaf Columns

A *Key* Leaf Column is a column that is a component of the primary key definition for certain objects on which it exists, such as the LEDGER_STAT table. This does not mean that the column is part of the primary key definition for all objects on which it exists. Rather, the column is part of the primary key for a specific list of objects identified by Table Classification.

A Non-key Leaf Column is never a component of such primary key definitions.

Registering a Leaf Column

Registering a new Leaf Column is a process. This process requires the following steps:

1. Add the column to required Objects
2. Re-register affected Objects
3. Modify Unique indexes (Key Leaf Columns only)
4. Assign the Processing Key Column Property (Key Leaf Columns only)
5. Register the Leaf Column

Each of these steps is discussed in detail.

> **Note:** FDM provides a utility script for performing the Add column and Re-register Objects steps (steps 1 and 2). This utility script eliminates the need for manually performing these tasks for any objects that are Tables. However, any registered objects that are Views still need to be re-created manually with the new Leaf Column. Refer to Chapter 21, "FDM Utilities" for more information.

Step 1: Adding the column to required Objects

FDM requires that Leaf Columns exist on a specific set of tables and views, based upon the Table Classification assignments in the database, prior to being able to register the new columns as a Leaf Column for the database. You can add the new column to these objects manually, or use the ADD_LEAF procedure. See Chapter 21, "FDM Utilities".

Adding the Column using the ADD_LEAF Procedure

FDM provides an ADD_LEAF procedure to automatically add the new Leaf Column to all required tables (views are not affected by this procedure - views must always be dropped and re-created manually to include the new column).

The utilities directory in the FDM server package includes supporting scripts for this procedure. The run_add_leaf script prompts you for the parameters required by the ADD_LEAF procedure. To execute this procedure, login to SQL*Plus as the FDM Schema Owner and type the following command from the **utilities/add_leaf** directory:

```
@run_add_leaf
```

This script prompts for the Leaf Column Name, Display Name, Leaf Type and DBF Name. Once valid parameters are entered, the script calls the ADD_LEAF procedure to add the new column as NUMBER(14) to all required tables (based upon Table Classification assignments). Refer to Chapter 17, "FDM Leaf Management"for more information regarding this script and the ADD_LEAF procedure.

Note: The ADD_LEAF procedure does not modify views. Manually drop and re-create any views with the designated Table Classification assignments so that they include the new Leaf Column.

Adding the Column Manually

Adding the new column manually involves executing Alter table commands for each table of the specified Table Classifications. For tables, use the Alter table add column syntax. For views, you must drop and re-create the view so that it has the new column in its definition.

The list of objects requiring the new column is based upon Table Classification assignments. For the Leaf Type = Both Instrument and Ledger, tables with the following Table Classification assignments are altered:

Table Classification CD	Description
50	Ledger Stat
200	TP Cash Flow
210	TP Non-Cash Flow

Table Classification CD	Description
300	Transaction Profitability
310	Instrument Profitability
351	All B Leaves
360	RM Standard
370	TP Option Costing

To identify the all of the objects for these Table Classifications, execute the following statement in SQL*Plus:

```
SELECT distinct(table_name)
FROM ofsa_table_class_assignment
WHERE table_classification_cd in (50, 200, 210, 300, 310, 351, 360, 370);
```

Note: The Leaf Objects reports provided with the FDM Discoverer Standard Reports displays the objects that must be modified for a column to be registered as an All Leaf or a Ledger Only Leaf.

For Leaf Columns of Registration Type Ledger Only, the Leaf Column must exist on each object registered for the following Table Classifications:

Table Classification CD	Description
352	All L Leaves

Execute the following statement in SQL*Plus to identify the objects to be modified:

```
SELECT distinct(table_name)
FROM ofsa_table_class_assignment
WHERE table_classification_cd in (352);
```

For each table identified by the SQL statements, add the new Leaf Column as NUMBER(14).

Note: Leaf Columns must be defined as NUMBER(14). FDM does not support columns of any other definition as Leaf Columns.

For tables, use the following syntax:

```
alter table table_name
add column column_name NUMBER(14);
```

For example, if you are adding a new TAX_ID column to the DEPOSITS table, use the following syntax:

```
alter table deposits
add column tax_id number(14);
```

For views, you must drop and re-create the view so that it includes the new column name as part of its definition.

Step 2: Re-register Objects

Once you have added the new column to all of the required objects, you must re-register these objects in FDM Administration. The ADD_LEAF procedure performs this step automatically for you (except for views), so you do not need to do it again if you used that procedure to add the column to your objects. If you did not use the ADD_LEAF procedure to add the column, you must manually Re-register the new column for all affected Objects. To do this, use the Re-register Object Wizard in FDM Administration to identify the new column for each object to which it was added in step 1. Assign the FDM Data Type LEAF to the new column for each object.

Note: If you used the ADD_LEAF procedure to add the new Leaf Column to your tables, you do not need to re-register any of the tables. However, you need to re-register any views (of the specified Table Classification assignments) because these were dropped and re-created manually in Step 1.

Note: The new column must be registered as FDM Data Type LEAF for all of the required objects.

For more information regarding Object Registration and Re-registration, refer to the *Oracle Financial Data Manager Administration Guide.*

Step 3: Modify Unique Indexes

For Leaf Columns designated of type *Key* FDM requires that the Leaf Column is added as a component of the unique index for objects registered for the following Table Classifications:

Table Classification CD	Description
350	Primary Key All Leaves

Use the following statement in SQL*Plus:

```
SELECT table_name, index_name
FROM user_indexes
WHERE uniqueness = 'UNIQUE'
AND table_name in
(SELECT C.table_name FROM ofsa_table_class_assignment C,
user_tab_columns U
WHERE C.table_name = U.table_name
AND U.column_name= :new_leaf
AND C.table_classification_cd = 350);
```

:new_leaf is the name of the new Leaf Column and must be in uppercase.

Drop the existing unique indexes for the identified tables, and re-create with the new column as an index component.

> **Caution:** The unique index for these tables must the same as the existing index, with only the new Leaf Column added as an additional component. Add the new Leaf Column as the last column name in the unique index. If the unique index is not re-created correctly for these tables, some OFS operations, such as Performance Analyzer Allocations, do not function properly.

Step 4: Assign the Processing Key Column Property

For Leaf Columns of type Key, the Processing Key Column Property needs to be associated to the new column for each object registered in step 3. Only those objects where the column was added to the unique index are affected. Add the Processing Key Column Property to the column for each of these objects.

For more information regarding adding Column Property assignments, refer to the *Oracle Financial Data Manager Administration Guide.*

Step 5: Register the Leaf Column

Once all of the requirements are met, the new column can be registered as a Leaf Column. To complete the registration process, use the Leaf Registration Wizard in FDM Administration.

For more information regarding the Leaf Registration Wizard, refer to the *Oracle Financial Data Manager Administration Guide.*

When you register a Leaf Column, a row is automatically inserted into OFSA_COLUMN_REQUIREMENTS for the column. This table identifies the required characteristics for the column. A row is also inserted into OFSA_PROPERTY_COLUMNS identifying the column as part of the Portfolio Table Classification.

Troubleshooting Leaf Registration

The Register - Leaf Column Wizard in FDM Administration validates that the specified Leaf column name is appropriately setup in the FDM database and Metadata. This validation checks the following conditions:

Leaf Column already identified as a Portfolio Column

FDM Administration verifies that the Leaf Column Name is not already entered in OFSA_COLUMN_REQUIREMENTS and OFSA_PROPERTY_COLUMNS as a Portfolio column.

OFSA_COLUMN_REQUIREMENTS stores the characteristics for Reserved columns (both FDM Reserved columns and user-defined Portfolio columns). The Leaf Registration Wizards attempts to insert the characteristics for the new column name into this table. OFSA_PROPERTY_COLUMNS identifies column names as Portfolio columns. If a record already exists in either OFSA_COLUMN_REQUIREMENTS or OFSA_PROPERT_COLUMNS for the new Leaf column name, FDM Administration reports an error.

In this situation, delete the existing entries in OFSA_COLUMN_REQUIREMENTS and OFSA_PROPERTY_COLUMNS for the specified column name. These existing records might have been inserted in a prior Leaf Registration for the specified column name (which was not completely undone), or it might be a designated Portfolio column. In either case, FDM Administration inserts a new entry for the column name during Leaf Registration, so it is safe to delete the existing entry.

Leaf Column is not registered as FDM Data Type LEAF

This error occurs when the Leaf Column is not registered correctly on all of the required objects. To identify which objects for which the Leaf column is not properly registered, execute one of the following SQL statements - depending upon the specified Table Classification list in the error message text:

For Ledger Only Leaves:

```
select table_name from ofsa_table_class_assignment
where table_classification_cd in (352)
and table_name not in (select table_name from ofsa_tab_columns
where column_name = leaf_column_name
and ofsa_data_type_cd = 10);
```

For B (All) Leaves:

```
select table_name from ofsa_table_class_assignment
where table_classification_cd in (50, 200, 210, 300, 310, 351, 360, 370)
and table_name not in (select table_name from ofsa_tab_columns
where column_name = leaf_column_name
and ofsa_data_type_cd = 10);
```

To resolve this error, register the Leaf Column on each identified object as FDM Data Type = LEAF in FDM Administration.

Column not part of the Process Key

This error occurs when the Leaf Column is not designated with the Processing Key Column Property for objects of the specified Table Classifications. To identify which objects are not assigned properly, execute the following SQL:

```
select table_name from ofsa_table_class_assignment
where table_classification_cd in (350)
and table_name not in (select table_name from ofsa_column_properties
where column_name = leaf_column_name
and column_property_cd = 15);
```

To resolve this error, assign the Processing Key Column Property to the specified Leaf Column name for each identified object in FDM Administration.

Unregistering a Leaf Column

Unregistering a Leaf Column does not unregister the column from any objects. Rather, it removes the identification of the column as being a Leaf Column. Once a Leaf Column is unregistered, it is not available for the Leaf Column operations within the OFS applications.

In order to unregister a Leaf Column, you first must remove any Processing Key Column Property assignments for the column name on any tables of Table Classification 350. To do this, use the Column Properties tab page within FDM Administration. The list of tables to be modified in this manner is the same list as in Step 4 (that is, the same objects for which the Processing Key Column Property assignment was modified).

Next, delete the entry from OFSA_COLUMN_REQUIREMENTS for the unregistered Leaf Column. OFSA_COLUMN_REQUIREMENTS stores the required FDM Data Types for all registered columns. Each Leaf Column has an entry in this table. When you unregister a Leaf Column, you need to remove it from this table so that FDM Administration does not default the FDM Data Type to LEAF for any newly registered columns of the same name. To do this, execute the following SQL statement:

```
delete from ofsa_column_requirements
where column_name = :leaf_column;
```

Once the column is deleted from this table, use the Unregister button in FDM Administration to unregister the Leaf Column.

> **Caution:** Prior to unregistering a Leaf Column, you must modify the unique indexes and Column Property assignments for the affected Objects as appropriate.

One additional step for unregistering a Leaf Column is to update the FDM Data Type for all objects on which the column is registered. The FDM Data Type should be updated to something other than LEAF. To do this, update the FDM Data Type field in the Columns tab under FDM Objects within the FDM Administration application.

Managing Leaf Values

A Leaf Value (also referred to as Leaf Node) is an individual entry for a Leaf Column. Leaf Values constitute the domain of valid values for any Leaf Column.

Users use the Leaf Setup screen within the OFS applications to add new Leaf Values for any Leaf Column.

Use the Synchronize Instruments procedure (described in Chapter 21, "FDM Utilities") to add a default Leaf Value entry for any unidentified entries in your Instrument objects. This procedure is particularly useful after you have loaded a new set of data into your Instrument tables.

18

FDM Database Performance Management

This chapter provides information about the duties of the Database Administrator to sustain the performance of the Oracle Financial Data Manager (FDM) database environment.

The following, specific topics are covered in this chapter:

- Tuning the FDM Database
- Performance Monitoring with BSTAT/ESTAT
- Index Management
- Managing Partitioned Tables and Indexes
- Rollback Segment Sizing and Management

Tuning the FDM Database

Tuning the FDM database can be a lengthy process, and, with many of the new Oracle features, also complex.

The tuning process, in general, follows these steps:

1. Identify the Oracle Financial Services Applications (OFSA) job types that your organization uses.

   The various job types are defined in the table in this section.

2. For each job type, time runs for a series of NumProcesses settings.

3. Based on the results, determine the appropriate setting per application.

It is recommended that for each process type, start with a NumProcesses setting of 1, and continue to double the setting until it is equal to the number of processors available on the application server for CPU bound jobs, and equal to the number of processors available on the database server for I/O bound jobs.

Generally, as the NumProcesses setting is increased, you can expect performance improvements up to a certain point. After that, processing times level off and then start increasing. Testing has shown that for all applications, the point at which processing time starts to increase occurs after the NumProcesses setting has exceeded the number of processors on the machine. Also, as the NumProcesses setting approaches the number of processors, the performance improvements are minimal.

> **Note:** To achieve optimal performance, you may need to change the NumProcesses settings for applications at different stages in the production cycle.

Special considerations need to be made when multiple OFSA jobs are run at the same time. Different types of jobs use resources differently. You may want to look into a scheduling tool to help you schedule dissimilar jobs to run at different times.

Besides the OFSA multiprocessing model, the database can be tuned for optimal performance by following these standard guidelines.

- Test the application and gather trace files for specific processes. Add and remove indexes as needed to support customizations.
- Review and modify initialization parameters for a precise database configuration. The primary initialization parameters used to tune the OFSA database are defined in Chapter 10, "FDM Database Installation".

- Partition primary (instrument) tables and indexes to enable I/O balancing.
- Review and modify default physical attributes (next extent, ini trans, free lists, degree, and so forth).

Tuning is usually a series of trade-offs. Once you have determined the bottlenecks, you may have to sacrifice in some areas to achieve your desired results. For example, if I/O is a problem, you may need to purchase more memory or more disks. If a purchase is not possible, you may have to limit the concurrency of the system. However, if you have clearly defined goals for performance, then the decision on what you need to relinquish to attain higher performance is simpler because you have identified the key areas to consider.

The following table lists the OFSA jobs by application and identifies whether the job is usually database bound or OFSA bound.

Application	Job Type	Generic Job Type	OFSA/DB Bound	MP Enabled	Comments
Balance & Control	Cash Flow Edits	Row by Row	DB or OFSA	Yes	If a large number of corrections need to be done, this may become OFSA bound
Balance & Control	Row by Row	Row by Row	DB or OFSA	Yes	As the number of rules in the correction processing ID increase, the process may become OFSA bound
Balance & Control	Bulk	Bulk	DB	Yes	
Performance Analyzer	Straight Ledger	Row by Row	DB	Yes	No percent or table ID. Ledger_Stat reading is MP enabled, Ledger_Stat writing is not.
Performance Analyzer	Percent Distribution	Row by Row	DB	Yes	
Performance Analyzer	Table ID	Row by Row	DB	Yes	
Performance Analyzer	Lookup Table ID	Bulk	DB	Yes	
Performance Analyzer	Detail	RBR/Bulk	DB	Yes	Row by Row for Ledger_Stat allocation, update to detail table is bulk
Transfer Pricing	Rate Migration	Bulk	DB	Yes	
Transfer Pricing	Bulk Transfer Pricing	Bulk	DB	Yes	
Transfer Pricing	Non-Cash Flow Transfer Pricing	Row by Row	DB	Yes	

Application	Job Type	Generic Job Type	OFSA/DB Bound	MP Enabled	Comments
Transfer Pricing	Cash Flow Transfer Pricing	Row by Row	OFSA	Yes	
Transfer Pricing	Ledger_Stat Migration	Row by Row	DB	Yes	
Risk Manager	Detail Processing (Current position, Gap, Market Value)	Row by Row	OFSA	Yes	All processing except Formula Leaves and Auto Balancing
Risk Manager	Formula Leaves	Row by Row	OFSA	No	
Risk Manager	Auto Balancing	Row by Row	OFSA	No	
Transformation	Ledger	Row by Row	DB	Yes	Dimension Filtering (aggregation) and complex OFSA filters affect performance
Transformation	Risk Manager	Row by row	DB	Yes	Dimension Filtering (aggregation) affects performance
Transformation	Roll up	Other	DB	No	

Performance Monitoring with BSTAT/ESTAT

One of the most important responsibilities for a DBA is ensuring that the Oracle database is properly tuned. Oracle RDBMS is highly tunable and enables the database to be monitored and adjusted to increase its performance. This section discusses how to make use of the utility called BSTAT/ESTAT to monitor performance within the Oracle database system.

Oracle provides two SQL scripts, called utlbstat.sql and utlestat.sql, that are used to capture a snapshot of system-wide, database performance statistics. These scripts have been renamed from BSTAT/ESTAT under Oracle6. In this chapter, the utility is referred to as BSTAT/ESTAT but the actual scripts are prefixed by utl.

On UNIX platforms, the scripts reside in the $ORACLE_HOME/rdbms/admin directory.

BSTAT Tables and Views

Utlbstat.sql creates a set of tables and views in your sys account. These contain a beginning snapshot of database performance statistics. The table names, which are listed in the following table, include the word *begin* to indicate beginning statistics.

View/Table Name	Description
stats$begin_dc	Dictionary Cache Statistics from v$rowcashe
stats$begin_event	System Wide Wait Statistics from v$system_event
stats$begin_file	Table of File I/O Statistics from stats$file_view
stats$begin_latch	Latch Statistics from v$latch
stats$begin_lib	Library Cache Statistics from v$librarycache
stats$begin_roll	Rollback Segment Statistics from v$rollstat
stats$begin_stats	General System Statistics from v$sysstat
stats$begin_view	View of File I/O Statistics from v$filestat, v$datafile, ts$, file$
stats$dates	Table containing beginning vdate and time

Additionally, utlbstat.sql creates a set of tables that contain the ending snapshot of database performance statistics. The table names, which are listed in the following table, include the word *end* to indicate ending statistics.

Table Name	Description
stats$end_d	Dictionary Cache Statistics from v$rowcache
stats$end_event	System Wide Wait Statistics from v$system_event
stats$end_file	Table of File I/O Statistics from stats$file_view
stats$end_latch	Latch Statistics from v$latch
stats$end_lib	Library Cache Statistics from v$librarycache
stats$end_roll	Rollback Segment Statistics from v$rollstat
stats$end_stats	General System Statistics v$sysstat

ESTAT Tables and Views

Utlestat.sql populates the end statistics tables and creates a set of tables in your sys account, which contain the difference between the beginning statistics and the ending statistics. The table names are listed as follows:

Table Name	Description
stats$dc	Dictionary Cache statistics
stats$event	System Wide Wait statistics
stats$files	File I/O statistics
stats$latches	Latch statistics
stats$lib	Library Cache statistics
stats$roll	Rollback Segment statistics
stats$stats	General System statistics
stats$dates	Table containing ending date and time

Finally, utlestat.sql creates a report in the current directory with the database performance statistics. The report is divided into the following sections:

- Library Cache Statistics
- System-Wide Statistics Totals
- System Event Statistics
- Average Length of Dirty Buffer Write Queue
- File I/O Statistics
- Tablespace I/O Statistic Totals
- Willing-To-Wait Latch Statistics
- No_Wait Latch Statistics
- Rollback Segment Statistics
- Init.ora Parameters
- Data Dictionary Cache Statistics
- Date/Time

Executing BSTAT/ESTAT

Complete the following steps to execute BSTAT/ESTAT:

1. Determine database activity to be monitored.

2. Move to the utlbstat.sql / utlestat.sql directory.
3. Log in to SQL*Plus as the SYS user.
4. Issue the following command:

   ```
   @utlbstat
   ```

5. Run application to be monitored.
6. Issue the following command:

   ```
   @utlestat
   ```

Run BSTAT / ESTAT only after your database has been running for a period of time. If you run BSTAT / ESTAT immediately after database startup, the buffer cache is not loaded and the statistics generated are not valid for database performance analysis.

If the database is shutdown in the middle of executing BSTAT / ESTAT the statistics are no longer valid because the V$ tables are initialized at shutdown and startup. If negative values are present for statistics other than Current_ statistics, the database has been shutdown and started during the execution of BSTAT / ESTAT.

In order for all of the statistics to be populated the init.ora parameter TIMED_ STATISTICS must be set to true. Setting this parameter causes slight performance degradation but is needed for the _Time_ statistics.

Depending on the type of applications that you are running, you may find it beneficial to start the database with different init.ora files. To determine if this is beneficial, run BSTAT/ESTAT for different applications. It is also beneficial to run BSTAT/ESTAT at different times of the day, both when the database is performing and when it is not performing. One set of statistics can then be used as a base and compared to other sets of statistics.

The following table provides two examples of when you might use BSTAT/ESTAT.

Methodology	Activity	Execution
Method 1	Several batch jobs are run every evening	Execute utlbstat.sql before the batch jobs are launched and execute utlestat.sql after the batch jobs are complete.
Method 2	Maximum processes access database from 1 - 4p.m.	Execute utlbstat.sql at 12:59p.m. and utlestat.sql at 4:01p.m.

The report generated when estat.sql is executed is called **report.txt** and is written to the current directory. This is the directory from which you executed SVRMGRL. Make sure that you have WRITE permissions to this directory or else the report will not be created and you will lose the statistics. This is because all the temporary tables would have been dropped when you ran estat.

Library Cache Statistics

Library Cache Statistics are generated from the table v$librarycache. The library cache contains shared SQL and PL/SQL areas.

The following table lists the column headers for this section of the report and a description of each header.

Column Header	Description
Library	Name of the Library cache namespace
Gets	Number of times the system request handles library objects belonging to this namespace
Gethitratio	Number of gethits divided by gets

Column Header	Description
Pins	Number of times an item in the library cache is executed
Pinhitratio	Number of pinhits divided by pins
Reload	Number of library cache misses on execution steps
Invalidations	Number of times that non-existent library objects (like shared SQL areas) have been invalidated

Library cache namespace can be one of the following:

- SQL AREA
- TABLE/PROCEDURE
- BODY
- TRIGGER
- INDEX
- CLUSTER
- OBJECT
- PIPE

Rows with the following namespace reflect library cache activity for SQL statements and PL/SQL blocks.

- SQL AREA
- TABLE/PROCEDURE
- BODY
- TRIGGER

Rows with any other namespace reflect library cache activity for object definitions that Oracle uses for dependency maintenance.

Gethitratio is the number of gethits divided by gets from v$librarycache where gethits are the number of times the handles are already allocated in the cache. Values close to 1 indicate that most of the handles the system has tried to get are cached.

Pins indicate the total number of times all SQL statements, PL/SQL blocks and object definitions were accessed for execution. A high pin rate is desirable.

Pinhitratio is the number of **pinhits** divided by **pins** from v$librarycache where pinhits are the number of times that objects the system is pinning and accessing are already allocated and initialized in the cache. Values close to 1 indicate that most of the objects the system has tried to pin and access are cached.

Reloads indicate the number of pins that resulted in library cache misses, causing Oracle to implicitly reparse a statement or block or reload an object definition because it had aged out of the library cache. An example of a reload is the number of times library objects have to be reinitialized and reloaded with data because they have been aged out or invalidated.

Total reloads should be near 0. If the ratio of reloads to pins is more than 1%, then you should reduce these library cache misses through allocating additional memory for the library cache. To increase the amount of memory available to the library cache, increase the value of SHARED_POOL_SIZE. To take advantage of the additional memory available for SQL areas, you may also need to increase OPEN_CURSORS.

> **Note:** Ensure that the increase in the SHARED_POOL_SIZE does not increase the SGA size to such an extent that it causes paging and swapping to occur on the machine.

System-Wide Statistics Totals

System-wide Statistic Totals are generated from the table v$sysstat, which contains general database system statistics.

The following table lists the column headers for this section of the report and a description of each header.

Column Header	Description
Statistic	Name of the system-wide statistic
Total	Total number of statistic operations
Per Trans	Total number of statistic operations / user commits
Per Logon	Total number of statistics operations per logon

Per Logon is always based on at least one logon because the estat script logs on as *internal*.

Outlined are several of the system-wide statistics that can be used to analyze database performance. Where possible, the init.ora parameters that influence these statistics are discussed.

DBWR Checkpoints

DBWR Checkpoints is the number of checkpoints messages that were sent to DBWR and not necessarily the total number of actual checkpoints that took place. During a checkpoint, there is a slight decrease in performance because data blocks are being written to disk, which causes I/O. If the number of checkpoints is reduced, the performance of normal database operations improves but recovery after instance failure is slower.

To reduce the number of checkpoints, increase the init.ora parameter LOG_CHECKPOINT_INTERVAL. If this parameter is set to a size (bytes) larger than the size of the redo log, a checkpoint is performed during each log switch. To increase the number of checkpoints and decrease database recovery time decrease the LOG_CHECKPOINT_INTERVAL parameter.

LOG_CHECKPOINT_TIMEOUT determines the amount of time to pass before the next checkpoint. Set this to 0 to disable time-based checkpoints.

Cluster Key Scan Block Gets/Scans

Cluster Key Scan Block Gets is the number of cluster blocks accessed. Cluster Key Scans is the number of scans processed on cluster blocks. If the ratio of Cluster Key Scan Block Gets to Cluster Key Scans is greater than one, the rows for one cluster key are stored in multiple data blocks and the cluster should be analyzed for row chaining.

The Size parameter specified during the Create Cluster command determines the number of cluster keys per block, with the default being one. If this parameter is not specified correctly, rows for one cluster key may not fit into one data block or there may be wasted space in the data block. If all of the data for one cluster key does not fit in one block, additional I/O must occur to access the data. See the *Oracle8i Administrator's Guide* to determine how to calculate the SIZE parameter for the Create Cluster command.

Consistent Gets and DB Block Gets

Consistent Gets is the number of blocks accessed in the buffer cache for queries without the select for update clause. DB Block Gets is the number of blocks accessed in the buffer cache for insert, update, delete and select for update operations. The sum of these is the total number of requests for data. This value

includes requests satisfied by access to buffers in memory. The formula is as follows:

Consistent Gets + DB Block Gets = Logical Reads

Physical Reads is the number of request for a block that caused a physical I/O. To calculate the hit ratio for the buffer cache use the following formula:

1 - (Physical Reads / Logical Reads) = Hit Ratio

If the hit ratio is lower than 70%, increase the init.ora parameter DB_BLOCK_BUFFERS. This increases the number of data block buffers in the SGA. Try to aim for a 90% + hit ratio.

Oracle can collect statistics that estimate the performance gain that would result from increasing the size of the buffer cache. These are collected in the table X$KCBRBH by setting the initialization parameter DB_BLOCK_LRU_EXTENDED_STATISTICS.

A large buffer may not always improve performance. On machines that have limited memory, too large a buffer may result in paging and swapping, which decreases overall performance. To analyze the effect of having a smaller buffer cache set the initialization parameter DB_BLOCK_LRU_STATISTICS and check the X$KCBCBH table. For further information on collecting and analyzing these statistics refer to "Tuning the Buffer Cache" in the *Oracle8i Administrator's Guide*.

Note: Ensure that any increase in DB_BLOCK_BUFFERS does not increase the SGA size to such an extent that it causes paging and swapping to occur on the machine.

Cumulative Opened Cursors

Cumulative Opened Cursors is the total open cursors that are opened during the execution of BSTAT/ESTAT. A cursor is opened for each SQL statement that is parsed into a context area. Performance is improved if cursors are reused because the SQL statements do not need to be reparsed. If a cursor is not reused, it is best to close the cursor when the SQL statement completes. Try to reduce the number of cursors opened per transaction as opposed to the total number opened during the BSTAT/ESTAT run. General guidelines are 5-7 per transaction for online processing and up to 10 for batch jobs.

To optimize cursor usage in the Oracle recompilers and SQL*Forms refer to the corresponding sections in *Oracle8i Designing and Tuning for Performance*.

Recursive Calls

Recursive Calls occur because of cache misses and segment extension. In general, if recursive calls is greater than 30 per process, the data dictionary cache should be optimized and segments rebuilt with storage clauses that result in fewer and larger extents. Segments include tables, indexes, rollback segment and temporary segments. Refer to *Oracle8i Designing and Tuning for Performance* for more information on optimizing the Data Dictionary Cache.

> **Note:** PL/SQL generates extra recursive calls, which may be unavoidable.

Redo Size

Use the following calculation to determine the average size of a redo entry:

Redo Size / Redo Entries = Average Size of Redo Entries

The redo size indicates how much redo was generated (in bytes) during the run and is useful in determining how large your redo log files should be.

Redo Log Space Requests

Redo Log Space Requests indicates how many times a user process waited for space in the redo log buffer. Try increasing the init.ora parameter LOG_BUFFER so that 0 Redo Log Space Requests are made.

> **Note:** LOG_BUFFER impacts the size of the SGA.

Redo Small Copies

Redo Small Copies is the total number of redo entries with fewer bytes than specified by the init.ora parameter LOG_SMALL_ENTRY_MAX_SIZE. These entries are written in the redo buffer under the protection of the redo allocation latch. If Redo Small Copies/Redo Entries is greater than 10%, the init.ora parameter LOG_SMALL_ENTRY_MAX_SIZE should be decreased to a size smaller than the average redo entry size. This reduces the number of redo entries copied on the redo allocation latch and enables more redo entries to be copied on the redo copy latch.

Table Scans

Table Scans on long tables is the total number of full table scans performed on tables with more than 5 db_blocks. If the number of full table scans is greater than 0 on a

per transaction the application should be tuned to effectively use Oracle indexes. Indexes should be used on long tables if more than 10% to 20% of the rows from the table are returned.

Table Scans on short tables is the number of full table scans performed on tables with less than 5 db_blocks. It is optimal to perform full table scans on short tables rather than using indexes. Table Scans on long tables plus Table Scans on short tables is equal to the number of full table scans performed during the execution of BSTAT / ESTAT.

Table Scan Blocks Gotten and Table Scan Rows Gotten respectively are the number of blocks and rows scanned during all full table scans. To determine, on average, the number of rows gotten per block for all full table scans, use the following formula:

Table Scan Rows Gotten / Table Scan Blocks Gotten

To determine the approximate number of blocks gotten for short and long table scans use the following formulas:

Table Scans (short) X 5 blocks = Blocks Scanned (short)

Table Scan Blocks Gotten - Blocks Scanned (short) = Blocks Scanned (long)

Blocks Scanned (long) / Table Scans (long tables) = Average number of blocks scanned per long table

Table Fetch by Rowid

Table Fetch by Rowid is the number of rows accessed by a rowid. This includes rows that are accessed using an index and rows that are accessed using direct rowid lookup. Rowid is the fastest path to a row and should be used whenever practical.

Table Fetch by Continued Row

Table Fetch by Continued Row indicates the number of rows that are chained to another block. In some cases, such as tables with long columns, this is unavoidable. However, the ANALYZE table command should be used to further investigate the chaining. Where possible, chaining should be eliminated by rebulding the table.

User Calls

User Calls is the number of times a call is made to the kernel. Parse Count indicates the number of times a SQL statement was parsed. The number of calls to the kernel should be reduced if possible. The *Performance Tuning Guide* tells you how to setup

array processing to reduce the number of calls to the kernel. Use the following calculation to determine the number of calls to the kernel per parse:

Parse Count / User Calls = Average calls per parse

System Event Statistics

This statistic is a system-wide collation of the per session wait statistics from v$session_events. The system-wide statistics are generated from the view v$system_event.

The following table lists the column headers for this section and a description of each header.

Column Header	Description
Column	Name of the event
Event Name	Total number of waits for the event
Total Time	Total time waited for event in hundredths of seconds
Average Time	Average time to wait

Average Time to wait for an event is calculated in hundredths of seconds, using the following formula:

total_time / count

Average Length of Dirty Buffer Write Queue

This statistic is generated from the view v$sysstat. The value is calculated from the following formula:

'summed dirty queue length' value / 'write requests' value

If this value is larger than the value of the _DB_BLOCK_WRITE_BATCH init.ora parameter, then consider increasing the value of _DB_BLOCK_WRITE_BATCH and check for disks that are doing more I/Os than other disks. You should also consider increasing the parameter DB_BLOCK_SIMULTANEOUS_WRITES.

File I/O Statistics

File I/O statistics are generated from the view stats$file_view.

The following table lists the column headers for this section and a description of each header.

Column Header	Description
Table_Space	Name of the data file's tablespace
File_Name	Name of the data file
Phys_Reads	Number of physical reads from the database file
Phys_Blks_Rd	Number of blocks read from the database file
Phys_Rd_Time	Time to read blocks (Timed_statistic must be set)
Phys_Writes	Number of physical writes to the database file
Phys_Blks_Wr	Number of physical blocks written to the database file
Phys_Wrt_Tm	Time to write blocks (Timed_statistic must be set)

File I/O should be spread evenly across multiple disk drives. In general, you should observe the following recommendation:

- Redo logs should be located on disks that do not contain database files
- Tables should be located on different disks than their associated indexes
- Large tables and indexes should be striped across several disks

- Active database files should not be located at opposite ends of the disk
- The most active database files should be located on the highest throughput disks.

The init.ora parameter DB_FILE_MULTI_BLOCK_READ_COUNT can be set to increase the number of blocks read during a single read. Increasing this parameter reduces I/O when full table scans are being performed.

Tablespace I/O Statistic Totals

These are generated from the view stat$file_view and are the same as File I/O statistics except that they are grouped by tablespace and therefore exclude the column file_name.

Willing-To-Wait Latch Statistics

Latch Statistics are generated from the view v$latch. The columns of this view reflect activity for different types of requests for latches. The columns GETS, MISSES and SLEEPS reflect willing-to-wait requests.

The following table lists the column headers for this section and a description of each header.

Column Header	Description
Name	Name of the latch
Gets	Number of successful willing-to-wait requests for a latch
Misses	Number of times an initial willing-to-wait request was unsuccessful
Hit_Ratio	Ratio of gets to misses
Sleeps	Number of times a process waited and requested a latch after an initial willing-to-wait request
Sleeps/Misses	Ratio of sleeps to misses

Hit_Ratio is calculated using the following formula:

(gets - misses)/gets

This value should be close to 1. If sleeps/misses > 1% then you are waiting multiple times for a latch.

No_Wait Latch Statistics

No_Wait Latch Statistics are generated from the view v$latch and the columns IMMEDIATE_GETS and IMMEDIATE_MISSES. A No_Wait or immediate latch does not wait for the latch to become free; it times out immediately.

The following table lists the column headers for this section and a description of each header.

Column Header	Description
Nowait_Gets	Name of the latch
Nowait_Misses	Number of successful immediate requests for each latch
Nowait_Ratio	Number of unsuccessful immediate requests for each latch

Nowait_Ratio is calculated using the following formula:

(nowait_gets - nowait_misses) / nowait_gets

This value should be close to 1.

Rollback Segment Statistics

Rollback Segment Statistics are generated from the view v$rollstat. The following table lists the column headers for this section and a description of each header.

Column Header	Description
Undo_Segments	Rollback segment number
Trans_Tbl_Gets	Number of gets for the rollback segment header
Trans_Tbl_Waits	Number of waits for the rollback segment header
Undo_Bytes_Wr	Number of bytes written to the rollback segment
Segment_Size_By	Size of the rollback segment in bytes
Xacts	Number of active transactions
Shrinks	Number of times the rollback segment shrank, eliminating one or more extents each time
Wraps	Number of time the rollback segment wraps from one extent to another

If the ratio of Trans_Tbl_Waits to Trans_Tbl_Gets is greater than 5%, additional rollback segments should be added to the database. In general, rollback segments should be the same size and created with a large number of small extents.

Init.ora Parameters

This section of the report contains a list of the init.ora parameters that were in effect during the execution of BSTAT/ESTAT. The output is generated from v$parameter where isdefault = 'FALSE'.

Data Dictionary Cache Statistics

The Data Dictionary Cache Statistics are generated from the table v$rowcache. The following table lists the column headers for this section and a description of each header.

Column Header	Description
Name	Dictionary cache name
Get_Reqs	Total number of requests for object
Get_Miss	Total number of object information not in cache
Scan_Req	Total number of scan requests
Scan_Miss	Total number of scan misses
Mod_Reqs	Number of inserts, updates and deletes
Count	Total number of entries in the cache
Cur_Usag	Total number entries for dictionary cache object

Misses on the data dictionary are to be expected in some cases. On instance startup, the data dictionary cache contains no data, so any SQL statement issued is likely to result in cache misses. To tune the cache, examine its activity only after your application has been running.

Examine cache activity by monitoring the ratio of GET_REQS to GET_MISS. For frequently accessed dictionary caches, the ratio of total GET_MISS to total GET_REQS should be less than 10% or 15%. If this ratio continues to increase you should consider increasing the amount of memory available to the data dictionary cache by increasing the value of SHARED_POOL_SIZE.

Date/Time

This section lists the date and time that BSTAT/ESTAT was executed.

Index Management

This section describes guidelines to follow when managing indexes and includes the following topics:

- Create Indexes After Inserting Table Data
- Limit the Number of Indexes per Table
- Specify the Tablespace for Each Index
- Specify Transaction Entry Parameters
- Specify Index Block Space Use
- Parallelize Index Creation
- Consider Creating UNRECOVERABLE Indexes
- Estimate Index Size and Set Storage Parameters
- OFSA-Specific Details
- Multiprocessing
- Updating Statistics
- Originally Supplied Indexes in FDM

Create Indexes After Inserting Table Data

Create an index for a table after inserting or loading data (using either SQL*Loader or Import) into the table. It is more efficient to insert rows of data into a table that has no indexes and then create the indexes for subsequent access. If you create indexes before the data is loaded, every index must be updated every time a row is inserted into the table.

When an index is created on a table that already has data, Oracle must use sort space. Oracle uses the sort space in memory allocated for the creator of the index (the amount per user is determined by the initialization parameter SORT_AREA_SIZE), but must also swap sort information to and from temporary segments allocated on behalf of the index creation.

To Manage a Large Index

If the index is extremely large, you may want to perform the following tasks:

1. Create a new temporary segment tablespace.
2. Alter the index creator's temporary segment tablespace.
3. Create the index.
4. Remove the temporary segment tablespace and re-specify the creator's temporary segment tablespace, if desired.

Under certain conditions, data can be loaded into a table with SQL*Loader's direct path load and an index created as data is loaded. Refer to the *Oracle8i Utilities* for more information on this methodology.

Limit the Number of Indexes per Table

A table can have any number of indexes. However, the more indexes there are, the more overhead is incurred as the table is modified. Specifically, when rows are inserted or deleted, all indexes on the table must be updated as well. Also, when a column is updated, all indexes that contain the column must be updated.

Thus, there is a trade-off between the speed of retrieving data from a table and the speed of updating the table. For example, if a table is primarily read-only, having more indexes can be useful; but if a table is heavily updated, having fewer indexes may be preferable.

Specify Transaction Entry Parameters

By specifying the INITRANS and MAXTRANS parameters during the creation of each index, you can affect how much space is allowed initially and the maximum space allocated for transaction entries in the data blocks of an index's segment.

For more information about setting these parameters, see the *Oracle8i Administrator's Guide.*

Specify Index Block Space Use

When an index is created for a table, data blocks of the index are filled with the existing values in the table up to PCTFREE. The space reserved by PCTFREE for an index block is only used when a new row is inserted into the table. The corresponding index entry must be placed in the correct index block (that is, between preceding and following index entries). If no more space is available in the appropriate index block, the indexed value is placed in another index block. Therefore, if you plan on inserting many rows into an indexed table, PCTFREE

should be high to accommodate the new index values. If the table is relatively static, without many inserts, PCTFREE for an associated index can be low so that fewer blocks are required to hold the index data.

PCTUSED cannot be specified for indexes. See the *Oracle8i Administrator's Guide* for additional information on this topic.

Specify the Tablespace for Each Index

Indexes can be created in the same or different tablespace as the table it indexes.

If you use the same tablespace for a table and its index, then database maintenance is more convenient (such as tablespace or file backup and application availability or update) and all the related data is online together.

Using different tablespaces (on different disks) for a table and its index produces better performance than storing the table and index in the same tablespace, due to reduced disk contention.

If you use different tablespaces for a table and its index and one tablespace is offline (containing either data or index), then the statements referencing that table are not guaranteed to work.

Parallelize Index Creation

If you have the parallel query option installed, you can parallelize index creation. Because multiple processes work together to create the index, Oracle can create the index more quickly than if a single server process created the index sequentially.

When creating an index in parallel, storage parameters are used separately by each query server process. Therefore, an index created with an INITIAL of 5 MB and a PARALLEL DEGREE of 12 consumes at least 60 MB of storage during index creation.

For more information on the parallel query option and parallel index creation, refer to *Oracle8i Designing and Tuning for Performance.*

Consider Creating UNRECOVERABLE Indexes

You can create an index without generating any redo log records by specifying UNRECOVERABLE in the CREATE INDEX statement.

Creating an unrecoverable index has the following benefits:

- Space is saved in the redo log files
- The time it takes to create the index is decreased

- Performance improves for parallel creation of large indexes

In general, the relative performance improvement is greater for larger unrecoverable indexes than for smaller ones. Creating small, unrecoverable indexes has little affect on the time it takes to create an index. However, for larger indexes the performance improvement can be significant, especially when you are also parallelizing the index creation.

Estimate Index Size and Set Storage Parameters

Estimating the size of an index before creating one is useful for two reasons. First, you can use the combined estimated size of indexes, along with estimates for tables, rollback segments and redo log files, to determine the amount of disk space required to hold an intended database. From these estimates, you can make your hardware purchases and other decisions relating to storage.

Second, you can use the estimated size of an individual index to better manage the disk space that the index uses. When an index is created, you can set appropriate storage parameters and improve I/O performance of applications that use the index.

For example, assume that you estimate the maximum size of a table before creating it. If you then set the storage parameters when you create the table, fewer extents are allocated for the table's data segment, and all of the table's data is stored in a relatively contiguous section of disk space. This decreases the time necessary for disk I/O operations involving this table.

The maximum size of a single index entry is approximately one-half the data block size minus some overhead.

As with tables, you can explicitly set storage parameters when creating an index. Try to store the index's data in a small number of large extents rather than a large number of small extents.

For specific information on storage parameters, see the *Oracle8i Administrator's Guide*. For specific information on estimating index size, see the same guide.

Considerations Before Disabling or Dropping Constraints

Because unique and primary keys have associated indexes, you should factor in the cost of dropping and creating indexes when considering whether to disable or drop a UNIQUE or PRIMARY KEY constraint. If the associated index for a UNIQUE key or PRIMARY KEY constraint is extremely large, you may save time by leaving the constraint enabled rather than dropping and re-creating the large index.

OFSA-Specific Details

The database upgrade process does not add or modify indexes on instrument tables. It is therefore recommended that the processes (SQL statements) that use these tables be analyzed.

You may find that some of the current indexes on leaves in instrument tables are not used or that new indexes should be created. It is important to understand which leaves are used, and how often the leaves are used in each WHERE clause. This may be different for each instrument table, and the indexes should reflect the level and type of leaf usage in each table. The performance of inserts, updates, and deletes is affected by each additional index. These indexing decisions should also consider the algorithm used by the database optimizer in executing a query.

Multiprocessing

Using multiprocessing can require additional indexes to the instrument tables (Ledger_Stat is not considered an instrument table). The flexibility included in OFSA 4.5 multiprocessing enables you to specify the columns used for data slicing and units of work. Because of this flexibility, it is the your responsibility to identify and create appropriate indexes for your processing needs.

The database upgrade process does not create any indexes on instrument and client data tables, because the application of these recommended indexes is dependent on the processing methodologies employed. This decision is dependent on your organization's use of the applications and data. To identify what indexes are needed, analyze the SQL statements generated during processing. See Chapter 19, "OFSA Multiprocessing" for more detailed information on configuring the FDM database for multiprocessing.

It is recommended that you concentrate on instrument tables, particularly joins involving these tables, and the WHERE/AND clauses in the SQL statements. A good starting point for this type of process is to analyze both the longest running processes and the processes executed most frequently. The SQL trace and TKPROF facilities can be used to capture SQL statements and analyze their EXPLAIN PLAN output. Refer to the *Oracle8i Designing and Tuning for Performance* for additional information on this topic.

Updating Statistics

Statistics should also be collected for instrument tables to aid the ORACLE optimizer in selecting an efficient execution plan for each SQL statement. In ORACLE, this is done by the ANALYZE command (see *Oracle8i SQL Reference*). The

ANALYZE command should be run for tables and their indexes after the initial data load and after running any process that either inserts or deletes a significant number of rows, or updates indexed columns.

Originally Supplied Indexes in FDM

For new FDM database installations, instrument tables are supplied initially, with the following indexes defined:

```
Unique Index : IDENTITY_CODE, ID_NUMBER (primary key index)
Secondary Non-Unique Index : ID_NUMBER
Secondary Non-Unique Index : AS_OF_DATE, ID_NUMBER
Secondary Non-Unique Index : COMMON_COA_ID, AS_OF_DATE
Secondary Non-Unique Index : ORG_UNIT_ID, AS_OF_DATE
Secondary Non-Unique Index : GL_ACCOUNT_ID, AS_OF_DATE
Secondary Non-Unique Index : ISO_CURRENCY_CD
```

The unique primary index should not be changed. As stated, other indexes are at your discretion. For example, if an instrument table contains data for only one month, the as_of_date column should be removed from each index for that table.

General Recommendations

Consider the following as general guidelines in tuning your database:

- Analyzing the longest running or most time-critical processes and corresponding SQL.
- Determine which org / application / [as_of_date] indexes are necessary for efficient multiprocessing.
- Create or drop indexes as necessary and periodically run the ANALYZE command on all instrument tables and their indexes to maximize performance.

Managing Partitioned Tables and Indexes

A partitioned table or index has been divided into a number partitions, which have the same logical attributes. For example, all partitions in a table share the same column and constraint definitions, and all partitions in an index share the same index options.

Each partition is stored in a separate segment and can have different physical attributes (such as PCTFREE, PCTUSED, INITRANS, MAXTRANS, TABLESPACE, and STORAGE).

Although you are not required to keep each table or index partition in a separate tablespace, it is to your advantage to do so. Storing partitions in separate tablespaces can:

- Reduce the possibility of data corruption in multiple partitions
- Make it possible to back up and recover each partition independently
- Make it possible to control the mapping of partitions to disk drives (important for balancing I/O load)

Creating Partitions

This section describes how to create table and index partitions.

Creating partitions is similar to creating a table or index - you must use the CREATE TABLE statement with the PARTITION CLAUSE. Also, you must specify the tablespace name for each partition when you have partitions in different tablespaces.

The following example shows a CREATE TABLE statement that contains 4 partitions, one for each sales quarter. A row with SALE_YEAR=1994, SALE_MONTH=7, and SALE_DAY=18 has the partitioning key (1994, 7, 18) and is in the third partition in the tablespace tsc. A row with SALE_YEAR=1994, SALE_MONTH=7, and SALE_DAY=1 has the partitioning key (1994, 7, 1), and also is in the third partition.

```
CREATE TABLE sales
(invoice_no NUMBER,
sale_year  INT NOT NULL,
sale_month INT NOT NULL,
sale_day   INT NOT NULL)
PARTITION BY RANGE
(sale_year, sale_month, sale_day)
(PARTITION sales_q1 VALUES LESS THAN (1994, 04, 01)
TABLESPACE tsa,
PARTITION sales_q2 VALUES LESS THAN (1994, 07, 01)
TABLESPACE tsb,
PARTITION sales_q3 VALUES LESS THAN (1994, 10, 01)
TABLESPACE tsc,
PARTITION sales q4 VALUES LESS THAN (1995, 01, 01)
TABLESPACE tsd);
```

Maintaining Partitions

This section describes the following partition maintenance operations:

- Moving Partitions
- Adding Partitions
- Dropping Partitions
- Truncating Partitions
- Splitting Partitions
- Merging Partitions
- Exchanging Table Partitions
- Rebuilding Index Partitions

For general information on partitioning, see *Oracle8i Concepts*.

For information on SQL syntax for DDL statements, see *Oracle8i SQL Reference*.

For information on catalog views that describe partitioned tables and indexes, and the partitions of a partitioned table or index, see the *Oracle8i Reference*.

For information on Import, Export and partitions, see *Oracle8i Utilities*.

Moving Partitions

You can use the MOVE PARTITION clause of the ALTER TABLE statement to perform the following:

- Re-cluster data and reduce fragmentation
- Move a partition to another tablespace
- Modify create-time attributes

Typically, you can change the physical storage attributes of a partition in a single step using an ALTER TABLE/INDEX MODIFY PARTITION statement. However, there are some physical attributes, such as TABLESPACE, that you cannot modify using MODIFY PARTITION. In these cases use the MOVE PARTITION clause.

Moving Table Partitions

You can use the MOVE PARTITION clause to move a partition. For example, you might want to move the most active partition to a tablespace that resides on its own disk, in order to balance I/O. To accomplish this you can issue the following statement:

```
ALTER TABLE parts MOVE PARTITION depot2
TABLESPACE ts094 NOLOGGING;
```

This statement always drops the partition's old segment and creates a new segment, even if you do not specify a new tablespace.

When the partition you are moving contains data, MOVE PARTITION marks the matching partition in each local index, and all global index partitions as unusable. You must rebuild these index partitions after issuing the MOVE PARTITION statement.

Moving Index Partitions

Some operations, such as MOVE PARTITION and DROP TABLE PARTITION, mark all partitions of a global index as unusable.

You can rebuild the entire index by rebuilding each partition individually using the ALTER INDEX REBUILD PARTITION statement. You can perform these rebuilds concurrently.

You can also drop the index and re-create it.

Adding Partitions

This section describes how to add new partitions to a partitioned table and how partitions are added to local indexes.

Adding Table Partitions

You can use the ALTER TABLE ADD PARTITION statement to add a new partition to the *high* end (the point after the last existing partition). If you want to add a partition at the beginning or in the middle of a table, or if the partition bound on the highest partition is MAXVALUE, use the SPLIT PARTITION statement instead.

When the partition bound on the highest partition is anything other than MAXVALUE, you can add a partition using the ALTER TABLE ADD PARTITION statement.

For example, if you have a table called SALES, which contains data for the current month in addition to the previous 12 months, you can add a partition for the current month using a statement similar to the one that follows.

In this example, on January 1, 1996, the DBA adds a partition for January.

```
ALTER TABLE sales
ADD PARTITION jan96
VALUES LESS THAN ('960201')
TABLESPACE tsx;
```

When there are local indexes defined on the table and you issue the ALTER TABLE... ADD PARTITION statement, a matching partition is also added to each local index. Because Oracle assigns names and default physical storage attributes to the new index partitions, you should consider renaming or altering them after the ADD operation is complete.

Adding Index Partitions

You cannot explicitly add a partition to a local index. Instead, new partitions are added to local indexes only when you add a partition to the underlying table.

You cannot add a partition to a global index because the highest partition always has a partition bound of MAXVALUE. If you want to add a new highest partition, use the ALTER INDEX SPLIT PARTITION statement.

Dropping Partitions

This section describes how to use the ALTER TABLE DROP PARTITION statement to drop table and index partitions and their data.

Dropping Table Partitions

You can use the ALTER TABLE DROP PARTITION statement to drop table partitions.

If local indexes are defined for the table, ALTER TABLE DROP PARTITION also drops the matching partition from each local index.

Dropping Table Partitions Containing Data and Global Indexes

If the partition contains data and global indexes, use either of the following methods to drop the table partition.

Method 1

Leave the global indexes in place during the ALTER TABLE DROP PARTITION statement. In this situation DROP PARTITION marks all global index partitions as unusable, so you must rebuild them afterwards.

The ALTER TABLE DROP PARTITION statement not only marks all global index partitions as unusable, but it also renders all non-partitioned indexes as unusable. Because the entire partitioned index cannot be rebuilt using one statement, **sal1**, in the following statement, is a non-partitioned index.

```
ALTER TABLE sales DROP PARTITION dec94;
ALTER INDEX sales_area_ix REBUILD sal1;
```

This method is best suited for large tables where the partition being dropped contains a significant percentage of the total data in the table.

Method 2

Issue the DELETE command to delete all rows from the partition before you issue the ALTER TABLE DROP PARTITION statement. The DELETE command updates the global indexes and also fires triggers and generates redo and undo logs.

Note: You can substantially reduce the amount of logging by setting the NOLOGGING attribute (using ALTER TABLE...MODIFY PARTITION...NOLOGGING) for the partition before deleting all of its rows.

For example, if you want to drop the first partition, which has a partition bound of 10000, you would issue the following statements:

```
DELETE FROM sales WHERE TRANSID < 10000;
ALTER TABLE sales DROP PARTITION dec94;
```

This method is best suited for small tables, or for large tables if the partition being dropped contains a small percentage of the total data in the table.

Dropping Table Partitions Containing Data and Referential Integrity Constraints

If a partition contains data and has referential integrity constraints, choose either of the following methods to drop the table partition.

Method 1

First, disable the integrity constraints, then issue the ALTER TABLE DROP PARTITION statement. Finally, enable the integrity constraints.

The following statement is an example of this method.

```
ALTER TABLE sales
DISABLE CONSTRAINT dname_sales1;
ALTER TABLE sales DROP PARTITTION dec94;
ALTER TABLE sales
ENABLE CONSTRAINT dname_sales1;
```

This method is best suited for large tables where the partition being dropped contains a significant percentage of the total data in the table.

Method 2

Issue the DELETE command to delete all rows from the partition before you issue the ALTER TABLE DROP PARTITION statement. The DELETE command enforces referential integrity constraints and also fires triggers and generates redo and undo logs. An example of this statement follows:

```
DELETE FROM sales WHERE TRANSID < 10000;
ALTER TABLE sales DROP PARTITION dec94;
```

This method is best suited for small tables, or for large tables if the partition being dropped contains a small percentage of the total data in the table.

Dropping Index Partitions

You cannot explicitly drop a partition from a local index. Local index partitions are dropped only when you drop a partition from the underlying table.

However, if a global index partition is empty, you can explicitly drop it by issuing the ALTER INDEX DROP PARTITION statement.

If a global index partition contains data, dropping the partition causes the next highest partition to be marked as unusable.

For example, if you want to drop the index partition P1, and P2 is the next highest partition, you must issue the following statements:

```
ALTER INDEX npr DROP PARTITION P1;
ALTER INDEX npr REBUILD PARTITION P2;
```

Note: You cannot drop the highest partition in a global index.

Truncating Partitions

Use the ALTER TABLE TRUNCATE PARTITION statement when you want to remove all rows from a table partition. You cannot truncate an index partition. However, the ALTER TABLE TRUNCATE PARTITION statement truncates the matching partition in each local index.

Truncating Partitioned Tables

You can use the ALTER TABLE TRUNCATE PARTITION statement to remove all rows from a table partition with or without reclaiming space. If there are local indexes defined for this table, ALTER TABLE TRUNCATE PARTITION also truncates the matching partition from each local index.

Truncating Table Partitions Containing Data and Global Indexes

If the partition contains data and global indexes, use either of the following methods to truncate the table partition.

Method 1

Leave the global indexes in place during the ALTER TABLE TRUNCATE PARTITION statement. In this situation TRUNCATE PARTITION marks all global index partitions as unusable, so you must use the ALTER INDEX REBUILD command.

The ALTER TABLE TRUNCATE PARTITION statement not only marks all global index partitions as unusable but it also renders all non-partitioned indexes as

unusable. Because the entire partitioned index cannot be rebuilt using one statement, **sal1**, in the following statement, is a non-partitioned index.

```
ALTER TABLE sales TRUNCATE PARTITION dec94;
ALTER INDEX sales_area_ix REBUILD sal1;
```

This method is best suited for large tables where the partition being truncated contains a significant percentage of the total data in the table.

Method 2

Issue the DELETE command to delete all rows from the partition before you issue the ALTER TABLE TRUNCATE PARTITION statement. The DELETE command updates the global indexes, and also fires triggers and generates redo and undo logs.

This method is best suited for small tables, or for large tables if the partition being truncated contains a small percentage of the total data in the table.

Truncating Table Partitions Containing Data and Referential Integrity Constraints

If a partition contains data and has referential integrity constraints, choose either of the following methods to truncate the table partition.

Method 1

First, disable the integrity constraints then issue the ALTER TABLE TRUNCATE PARTITION statement. Finally, enable the integrity constraints. The following statement is an example of this method.

```
ALTER TABLE sales
DISABLE CONSTRAINT dname_sales1;
ALTER TABLE sales TRUNCATE PARTITTION dec94;
ALTER TABLE sales
ENABLE CONSTRAINT dname_sales1;
```

This method is best suited for large tables where the partition being truncated contains a significant percentage of the total data in the table.

Method 2

Issue the DELETE command to delete all rows from the partition before you issue the ALTER TABLE TRUNCATE PARTITION statement. The DELETE command enforces referential integrity constraints, and also fires triggers and generates redo and undo logs.

You can substantially reduce the amount of logging by setting the NOLOGGING attribute (using ALTER TABLE...MODIFY PARTITION...NOLOGGING) for the partition before deleting all of its rows. An example of this statement follows:

```
DELETE FROM sales WHERE TRANSID < 10000;
ALTER TABLE sales TRUNCATE PARTITION dec94;
```

This method is best suited for small tables, or for large tables if the partition being truncated contains a small percentage of the total data in the table.

Splitting Partitions

This form of ALTER TABLE/INDEX divides a partition into two partitions. You can use the SPLIT PARTITION clause when a partition becomes too large and causes backup, recovery or maintenance operations to take a long time. You can also use the SPLIT PARTITION clause to redistribute the I/O load.

Splitting Table Partitions

You can split a table partition by issuing the ALTER TABLE SPLIT PARTITION statement. If there are local indexes defined on the table, this statement also splits the matching partition in each local index. Because Oracle assigns system-generated names and default storage attributes to the new index partitions, you should consider renaming or altering these index partitions after splitting them.

If the partition you are splitting contains data, the ALTER TABLE SPLIT PARTITION statement marks the matching partitions (there are two) in each local index, as well as all global index partitions, as unusable. You must rebuild these index partitions after issuing the ALTER TABLE SPLIT PARTITION statement.

Splitting a Table Partition: Scenario

This example describes the method for splitting a table partition.

In this scenario, fee_katy is a partition in the table VET_cats, which has a local index, JAF1. There is also a global index, VET on the table. VET contains two partitions, VET_parta, and VET_partb.

To split the partition fee_katy and rebuild the index partitions, you need to issue the following statements:

```
ALTER TABLE vet_cats SPLIT PARTITION
fee_katy at (100) INTO ( PARTITION
fee_katy1 ..., PARTITION fee_katy2 ...);


ALTER INDEX JAF1 REBUILD PARTITION SYS_P00067;
ALTER INDEX JAF1 REBUILD PARTITION SYS_P00068;
ALTER INDEX VET REBUILD PARTITION VET_parta;
ALTER INDEX VET REBUILD PARTITION VET_partb;
```

You must examine Oracle's system tables to locate the names assigned to the new, local index partitions. In this scenario, they are SYS_P00067 and SYS_P00068. If you want, you can rename them.

Also, unless JAF1 already contained the partitions fee_katy1 and fee_katy2, names assigned to local index, partitions produced by this split will match those of the corresponding base table partitions.

Splitting Index Partitions

You cannot explicitly split a partition in a local index. A local index partition is split only when you split a partition in the underlying table.

You can issue the ALTER INDEX SPLIT PARTITION statement to split a partition in a global index if the partition is empty.

The following statement splits the index partition containing data, QUON1:

```
ALTER INDEX quon1 SPLIT
PARTITION canada AT VALUES LESS THAN ( 100 ) INTO
PARTITION canada1 ..., PARTITION canada2 ...);
ALTER INDEX quon1 REBUILD PARTITION canada1;
ALTER INDEX quon1 REBUILD PARTITION canada2;
```

Merging Partitions

While there is no explicit MERGE statement, you can merge a partition using either the DROP PARTITION or EXCHANGE PARTITION clauses.

Merging Table Partitions

You can use either of the following methods to merge table partitions.

Method 1

If you have data in partition OSU1 and no global indexes or referential integrity constraints on the table, OH, you can merge table partition OSU1 into the next highest partition, OSU2.

To merge partition OSU1 into partition OSU2 complete the following steps:

1. Export the data from OSU1.
2. Issue the following statement:

```
ALTER TABLE OH DROP PARTITION OSU1;
```

3. Import the data from Step 1 into partition OSU2.

Note: The corresponding local index partitions are also merged

Method 2

Another way to merge partition OSU1 into partition OSU2 is to complete the following steps:

1. Exchange partition OSU1 of table OH with *dummy* table COLS.
2. Issue the following statement:

```
ALTER TABLE OH DROP PARTITION OSU1;
```

3. Insert as SELECT from the *dummy* table to move the data from OSU1 back into OSU2.

Merging Partitioned Indexes

The only way to merge partitions in a local index is to merge partitions in the underlying table.

If the index partition BUCKS is empty, you can merge global index partition BUCKS into the next highest partition, GOOSU, by issuing the following statement:

```
ALTER INDEX BUCKEYES DROP PARTITION BUCKS;
```

If the index partition BUCKS contains data, then issue the following statements:

```
ALTER INDEX BUCKEYES DROP PARTITION BUCKS;
ALTER INDEX BUCKEYES REBUILD PARTITION GOOSU;
```

While the first statement marks partition GOOSU as unusable, the second makes it valid again.

Exchanging Table Partitions

You can convert a partition into a non-partitioned table, and convert a table into a partition of a partitioned table by exchanging their data (and index) segments. Exchanging table partitions is most useful when you have an application using non-partitioned tables that you want to convert to partitions of a partitioned table. For example, you may already have partition views that you want to migrate into partitioned tables.

Merging Adjacent Table Partitions: Scenario

This example describes how to merge two adjacent table partitions.

Suppose you want to merge two partitions, FEB95 and MAR95, of the SALES table by moving the data from the FEB95 partition into the MAR95 partition.

To merge the two table partitions complete the following steps:

1. Create a temporary table to hold the FEB95 partition data using statements similar to the following:

   ```
   CREATE TABLE sales_feb95 (...)
   TABLESPACE ts_temp STORAGE (INITIAL 2);
   ```

2. Exchange the FEB95 partition segment into the table SALES_FEB95, using the following statement:

   ```
   ALTER TABLE sales
   EXCHANGE PARTITION feb95 WITH TABLE
   sales_feb95 WITHOUT VALIDATION;
   ```

   Now the SALES_FEB95 table placeholder segment is attached to the FEB95 partition.

3. Drop the FEB95 partition by issuing the following statement:

   ```
   ALTER TABLE sales DROP PARTITION feb95;
   ```

   This frees the segment originally owned by the SALES_FEB95 table.

4. Move the data from the SALES_FEB95 table into the MAR95 partition using an INSERT statement:

```
INSERT INTO sales PARTITION (mar95)
SELECT * FROM sales_feb95;
```

   Using the extended table name in this situation is more efficient. Instead of attempting to compute the partition to which a row belongs, Oracle verifies that it belongs to the specified partition.

5. Drop the SALES_FEB95 table to free the segment originally associated with the FEB95 partition, using the following statement:

```
DROP TABLE sales_feb95;
```

6. Rename the MAR95 partition (this step is optional).

```
ALTER TABLE sales RENAME PARTITION mar95 TO feb_mar95;
```

For more information on deferring index maintenance, see the ALTER SESSION SET SKIP_UNUSABLE_INDEXES statement in *Oracle8i SQL Reference.*

Rebuilding Index Partitions

Some operations, such as ALTER TABLE DROP PARTITION, mark all partitions of a global index as unusable. You can rebuild global index partitions in either of the following ways:

- Rebuild each partition by issuing the ALTER INDEX REBUILD PARTITION statement (you can run the rebuilds concurrently).
- Drop the index and re-create it.

  This method is more efficient because the table is scanned only once.

Rollback Segment Sizing and Management

The total rollback segment size should be set based on the size of the most common transactions issued against a database. In general, short transactions experience better performance when the database has many smaller rollback segments, while long-running transactions, like OFSA batch jobs, perform better with larger rollback segments. Generally, rollback segments can handle transactions of any size easily; however, in extreme cases when a transaction is either very short or very long, a user might want to use an appropriately sized rollback segment.

If a system is running only short transactions, rollback segments should be small so that they are always cached in main memory. If the rollback segments are small

enough, they are more likely to be cached in the SGA according to the LRU algorithm, and database performance is improved because less disk I/O is necessary.

The main disadvantage of small rollback segments is the increased likelihood of the error *snapshot too old* when running a long query involving records that are frequently updated by other transactions. This error occurs because the rollback entries needed for read consistency are overwritten as other update entries wrap around the rollback segment. Consider this issue when designing an application's transactions, and make them short atomic units of work so that you can avoid this problem.

In contrast, long running transactions work better with larger rollback segments, because the rollback entries for a long running transaction can fit in pre-allocated extents of a large rollback segment.

When a database system's applications concurrently issue a mix of very short and very long transactions, performance can be optimized if transactions are explicitly assigned to a rollback segment based on the transaction/rollback segment size.

You can also minimize dynamic extent allocation and truncation for rollback segments. This is not required for most systems and is intended for extremely large or small transactions. To optimize performance when issuing a mix of extremely small and large transactions, make a number of rollback segments of appropriate size for each type of transaction (such as small, medium and large). Most rollback segments should correspond to the typical transactions, with a fewer number of rollback segments for the atypical transactions. Then set OPTIMAL for each such rollback segment so that the rollback segment returns to its intended size if it has to grow.

You should tell end users about the different sets of rollback segments that correspond to the different types of transactions. Often, it is not beneficial to assign a transaction explicitly to a specific rollback segment. However, you can assign an atypical transaction to an appropriate rollback segment created for such transactions. For example, you can assign a transaction that contains a large batch job to a large rollback segment.

When a mix of transactions is not prevalent, each rollback segment should be 10% of the size of the database's largest table. This is because most SQL statements affect 10% or less of a table; therefore, a rollback segment of this size should be sufficient to store the actions performed by most SQL statements.

In general, you should set a high MAXEXTENTS for rollback segments. This provides a rollback segment to allocate subsequent extents, as needed.

Create Rollback Segments with Many Equally Sized Extents

Each rollback segment's total allocated space should be divided among many equally sized extents. In general, optimal rollback I/O performance is observed if each rollback segment for an instance has 10 to 20 equally sized extents.

After determining the desired total initial size of a rollback segment and the number of initial extents for the segment, use the following formula to calculate the size of each extent of the rollback segment:

T / n = s

where:

T = total initial rollback segment size, in bytes

n = number of extents initially allocated

s = calculated size, in bytes, of each extent initially allocated

After **s** is calculated, create the rollback segment and specify the storage parameters INITIAL and NEXT as **s**, and MINEXTENTS to **n**. PCTINCREASE cannot be specified for rollback segments and therefore defaults to 0. Also, if the size **s** of an extent is not an exact multiple of the data block size, it is rounded up to the next multiple.

Set an Optimal Number of Extents for Each Rollback Segment

You should carefully assess the kind of transactions the system runs when setting the OPTIMAL parameter for each rollback segment. For a system that executes long-running transactions frequently, OPTIMAL should be large so that Oracle does not have to shrink and allocate extents frequently. Also, for a system that executes long queries on active data, OPTIMAL should be large to avoid *snapshot too old* errors. OPTIMAL should be smaller for a system that mainly executes short transactions and queries, so that the rollback segments remain small enough to be cached in memory, thus improving system performance.

19

OFSA Multiprocessing

This chapter provides information on configuring the Oracle Financial Services Application (OFSA) server-centric software for multiprocessing.

The following topics are covered in this chapter:

- Multiprocessing Model
- Multiprocessing Options
- Specifying Multiprocessing Parameters
- Tuning Multiprocessing
- Migration from OFSA 3.5/4.0
- Examples

Caution: OFS application multiprocessing settings in FDM 4.5 are no longer specified in the server ini files. Rather, they are designated in the database. Because of this, all OFS application multiprocessing settings revert to the default after the FDM upgrade process is complete. If you are upgrading to FDM 4.5 from a previous release of OFSA, refer to Chapter 11, "Upgrading from OFSA 3.5/4.0"for more information.

Multiprocessing Model

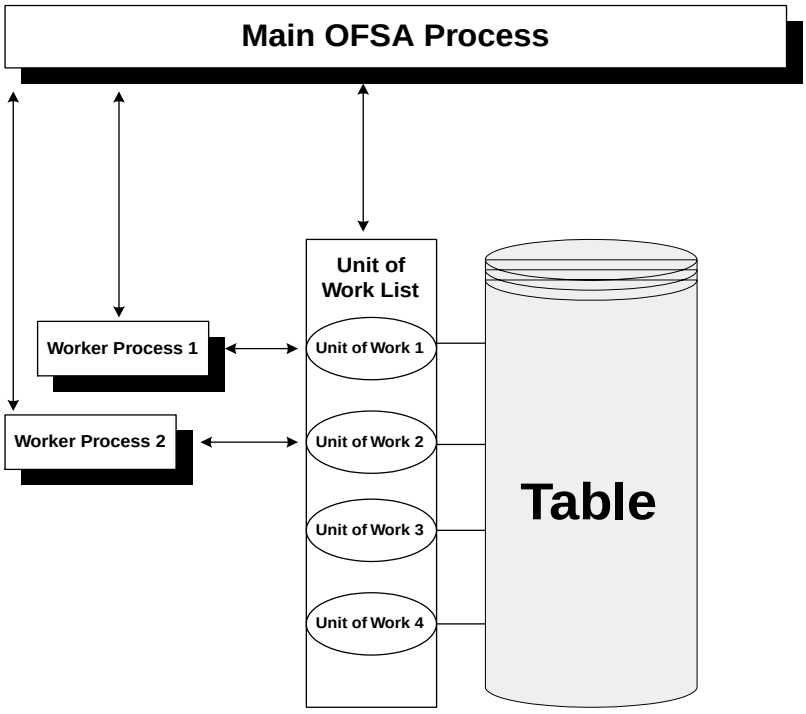
By default multiprocessing is disabled for all OFS applications. Multiprocessing is enabled by setting application specific parameters located in tables within the Oracle Financial Data Manager (FDM) data model. The following applications and features have multiprocessing settings:

- Oracle Balance & Control – Data Correction Process ID
- Customer Householding
- Performance Analyzer – Allocation ID
- Risk Manager – Risk Manager Process ID
- Transfer Pricing – Transfer Pricing Process ID
- Transformation ID

OFSA multiprocessing is based on the concept of a *unit of work*. A *unit of work* is a set of rows from the database. A single OFSA process becomes multiple processes by dividing the single process according to distinct sets of rows. *Units of work* are distributed to worker processes until all processes have been completed. To achieve multiple parallel processes, the following options must be configured:

- creating a list(s) of units of work
- defining the number of worker processes to service the units of work lists
- defining how the worker processes service the unit of work lists

The specifics of each option are discussed. However, the following diagram illustrates the basic multiprocessing principles:



1. The main process makes a list of all units of work that need to be processed.
2. The main process spawns worker processes. Each worker process is assigned a unit of work by the main process.
3. When all units of work have been completed, the worker process exits and the main process finishes any clean-up aspects of processing.
4. During processing the following is true:
   - Each worker process must form its own database connection.
   - A unit of work is processed only by a single worker process.

- Different units of work are processed at the same time by different worker processes.

Note: If data is not distributed well across physical devices, I/O contention may offset the advantage of parallelism within OFSA for I/O bound processing.

Multiprocessing Options

The Multiprocessing Options are the settings and parameters that control how individual OFSA IDs are processed by the OFSA engines. The FDM database includes default settings for all of the multiprocessing options. However, you can also customize the settings for your own use. The section describes the different Multiprocessing options as well as how to customize each. These options are:

- Units of Work
- Unit of Work Servicing
- Worker Processes

Units of Work

The OFSA processing engines determine units of work for any job based upon the Process Data Slicing Code (PROCESS_DATA_SLICES_CD) assignment. The Data Slicing Code is comprised of one or more columns by which data in the (processed) table is segmented. The individual segments are the defined Units of Work.

The OFSA multiprocessing model enables you to specify different Unit of Work definitions for your processes. You could specify one Unit of Work definition for one set of processes, and then specify a different Unit of Work definition for another set of processes.

The OFSA Processing Engines determine the units of work for a job by executing the following statement (with filtering criteria applied) on every table the process is run against:

```
select distinct <data slice columns> from <table>
where <filter condition>;
```

where <data slice columns> is the comma-separated list of columns used for data slicing, <table> is the name of the table being processed and <filter condition> is the additional filter (if any) for the process. Any column(s) in a table can be used for data slicing.

Default Unit of Work Definitions

OFSA provides three default Unit of Work definitions:

PROCESS_DATA_SLICES_CD	PROCESS_DATA_SLICES_SEQ	COLUMN_NAME
1	1	ORG_UNIT_ID
1	2	COMMON_COA_ID
2	1	ORG_UNIT_ID
3	1	COMMON_COA_ID

Any single Process Data Slice Code can be comprised of multiple columns. As an example of this, the PROCESS_DATA_SLICES_CD = 1 is comprised of both ORG_UNIT_ID and COMMON_COA_ID. The PROCESS_DATA_SLICES_SEQ identifies the precedence for the columns within the Process Data Slices CD.

Creating Customized Unit of Work Definitions

In order to create a customized Unit of Work definition, you need to create a new PROCESS_DATA_SLICES_CD value and specify appropriate parameters for it.

OFSA_PROCESS_DATA_SLICES and OFSA_PROCESS_DATA_SLICES_DTL tables control the data slice columns and the resulting order of units of work. Data slicing methods are created by inserting a new code value into OFSA_PROCESS_DATA_SLICES.PROCESS_DATA_SLICES_CD. Similarly, the columns used for data slicing are created by inserting new rows into OFSA_PROCESS_DATA_SLICES_DTL.

The descriptions for the columns in OFSA_PROCESS_DATA_SLICES and OFSA_PROCESS_DATA_SLICES_DTL are provided for your reference.

TABLE_NAME	COLUMN_NAME	DISPLAY_NAME	DESCRIPTION
OFSA_PROCESS_DATA_SLICES	PROCESS_DATA_SLICES_CD	Process Data Slices Code	Process Data Slices Code
OFSA_PROCESS_DATA_SLICES_DTL	PROCESS_DATA_SLICES_CD	Process Data Slices Code	Process Data Slices Code

OFSA_PROCESS_DATA_SLICES_DTL	PROCESS_DATA_SLICES_SEQ	Process Data Slices Sequence	Precedence of slicing the data
OFSA_PROCESS_DATA_SLICES_DTL	COLUMN_NAME	Column Name	Column name used for slicing

In order to create a customized Unit of Work definition, you need to insert data into the OFSA_PROCESS_DATA_SLICES and OFSA_PROCESS_DATA_SLICES_DTL tables. Example data illustrating a customized Unit of Work definition is as follows:

OFSA_PROCESS_DATA_SLICES

PROCESS_DATA_SLICES_CD
4

OFSA_PROCESS_DATA_SLICES_DTL

PROCESS_DATA_SLICES_CD	PROCESS_DATA_SLICES_SEQ	COLUMN_NAME
4	1	ORG_UNIT_ID
4	2	TP_COA_ID

Unit of Work Servicing

Unit of Work Servicing identifies how the OFSA processing engines interact with Oracle RDMBS Table Partitioning.

What is Partitioning?

Partitioning addresses the key problem of supporting very large tables and indexes by enabling you to decompose them into smaller and more manageable pieces called partitions. Once partitions are defined, SQL statements can access and manipulate the partitions rather than entire tables or indexes. Partitions are especially useful in data warehouse applications, which commonly store and analyze large amounts of historical data. See *Oracle 8i Concepts* for more information.

What is Unit of Work Servicing?

Unit of Work Servicing specifies how individual units of work are processed for a table that is partitioned.

For a partitioned table, an application Process ID can create multiple Units of Work Lists by executing the following statement (with filtering criteria applied) on every table partition the process is run against:

select distinct <data slice columns> from <table_partition_n>

where <data slice columns> is the comma-separated list of columns used for data slicing. Any column(s) in a table can be used for data slicing. <table_partition_n> are the unique table partitions of a table where n is assumed to be greater than 1.

The different Servicing methodologies are stored in the OFSA_PROCESS_ PARTITION and OFSA_PROCESS_PARTITION_MLS tables. You cannot add any customized Servicing methodologies. The Servicing methodologies provided in Release 4.5 are listed as follows:

PROCESS_PARTITION_CD	PROCESS_PARTITION
0	Do not use partitions (Single Servicing)
1	Use shared partitions (Cooperative Servicing)
2	Use non-shared partitions (Dedicated Servicing)

These methodologies are defined as follows:

- Single Servicing
- Cooperative Servicing
- Dedicated Servicing

Single Servicing

Single Servicing indicates that the OFSA processing engine fulfils unit of work requests regardless of any table partitioning. As each individual process completes, it requests the next unit of work segment, whether or not that segment belongs in the same Table partition.

Use Single Servicing when you do not have Oracle Table Partitioning in your database.

Cooperative Servicing

Cooperative Servicing indicates that the OFSA processing engine fulfils unit of work requests so that each process works against a specific partition unless it is idle. Idle processes then work against the next available unit of work segment, whether or not that segment belongs in the same Table partition.

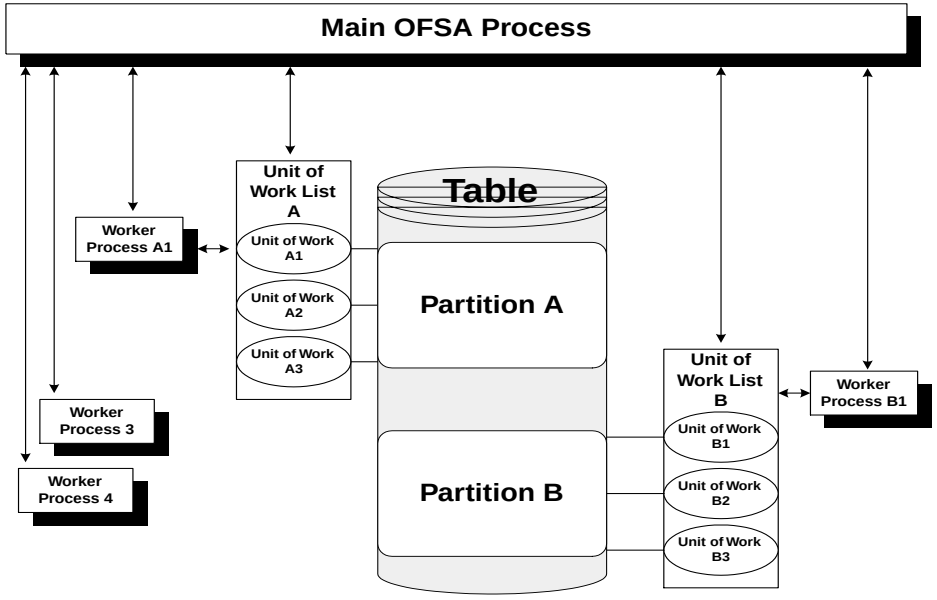
Dedicated Servicing

Dedicated Servicing indicates that the OFSA processing engine fulfils unit of work requests so that each process works against a specific partition.

Examples of How Worker Processes Service Units of Work

OFSA_PROCESS_ID_STEP_RUN_OPT.PROCESS_PARTITION_CD defines how Worker Processes service the Units of Work Lists(s). As explained in the Define Units of Work List(s) step, a OFSA_PROCESS_ID_STEP_RUN_OPT.PROCESS_ PARTITION_CD equal to 0 results in a single Units of Work List. With a single Units of Work List, all available worker processes service the list until all Units of Work are complete. When OFSA_PROCESS_ID_STEP_RUN_OPT.PROCESS_ PARTITION_CD equals 1 or 2 and the table to be processed is partitioned, multiple Units of Work Lists are created. The following scenarios explain how the worker processes service multiple Units of Work Lists:

Scenario 1: Number of Worker Processes > Number of Units of Work Lists

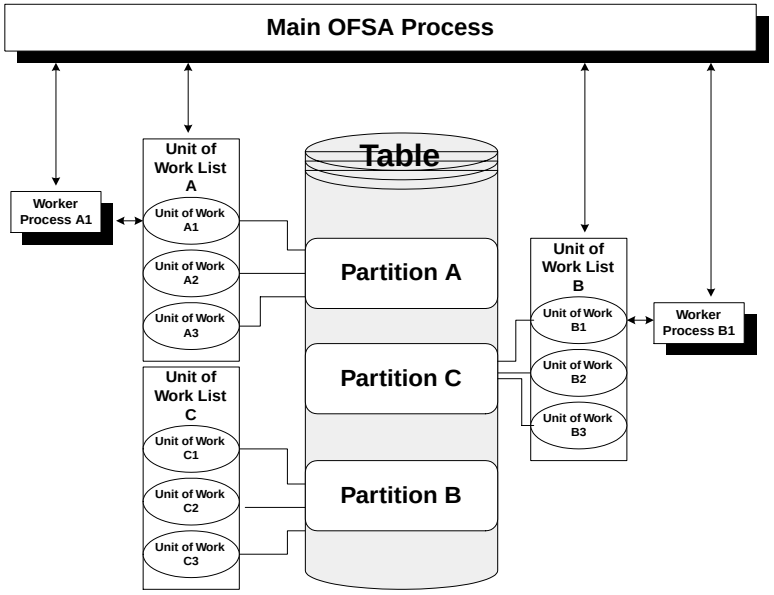


- The main process makes two lists of all units of work that need to be processed, Unit of Work List A and Unit of Work List B respectively. (The setup is that the Table has two partitions.)

- The main process spawns four worker processes. A dedicated worker process is assigned to service each Units of Work List, Worker Process A1 and Worker Process B1 respectively. (The setup is (OFSA_PROCESS_ID_STEP_RUN_OPT.NUM_OF_PROCESSES = 4)
  - If OFSA_PROCESS_ID_STEP_RUN_OPT.PROCESS_PARTITION_CD equals 1, Worker Process 3 and Worker Process 4 assist Worker Process A1. When a Unit of Work List is complete the available worker processes assist dedicated worker process on their Unit of Work List.
  - If OFSA_PROCESS_ID_STEP_RUN_OPT.PROCESS_PARTITION_CD equals 2, Worker Process 3 and Worker Process 4 do not assist the dedicated worker processes.
- When all units of work have been completed, the worker process exits and the main process finishes any clean-up aspects of processing.
- During processing the following is true:
  - Each worker process must form its own database connection.
  - A unit of work is processed only by a single worker process.
  - Different units of work are processed at the same time by different worker processes.

Scenario 2: Number of Worker Processes < Number of Units of Work Lists

- The main process makes three lists of all units of work that need to be processed, Unit of Work List A, Unit of Work List B, and Unit of Work List C respectively. (The setup is that the Table has three partitions.)
- The main process spawns two worker processes. A dedicated worker process is assigned to service a Units of Work List, Worker Process A1 and Worker Process B1 respectively. (The setup is (OFSA_PROCESS_ID_STEP_RUN_OPT.NUM_OF_PROCESSES = 2)
  - If OFSA_PROCESS_ID_STEP_RUN_OPT.PROCESS_PARTITION_CD equals 1, Worker Process A1 and Worker Process B1 work until all units of work are complete from all three Unit of Work Lists.
  - If OFSA_PROCESS_ID_STEP_RUN_OPT.PROCESS_PARTITION_CD equals 2, the first worker process to complete their Unit of Work List services Unit of Work List C. When the other worker process completes their list, the worker process exits.



- When all units of work have been completed, the worker process exits and the main process finishes any clean-up aspects of processing.
- During processing the following is true:
  - Each worker process must form its own database connection.
  - A unit of work is processed only by a single worker process.
  - Different units of work are processed at the same time by different worker processes.

Worker Processes

Worker Processes is the number of individual processes working simultaneously to complete the job. The Main OFSA Process launches the individual worker processes. OFSA enables you to specify the number of worker processes for your jobs.

Specifying Multiprocessing Parameters

The FDM Database Creation and Database Upgrade processes seed default multiprocessing parameters. By default, multiprocessing is turned off for all

processes. In order to turn multiprocessing on, you must specify the multiprocessing parameters for your jobs.

This section discusses the following topics:

- Multiprocessing Assignment Levels
- Defining Multiprocessing
- Engine Overrides

Multiprocessing Assignment Levels

Multiprocessing parameters can be specified at different levels. A Multiprocessing Assignment Level is the category of OFSA IDs that are processed with a designated set of multiprocessing parameters.

OFSA provides multiprocessing assignments at the following levels:

- Processing Engine
- Processing Engine Step
- OFSA ID

Processing Engine

When specifying multiprocessing parameters at the Processing Engine level, all Processing IDs for that engine are processed with the designated parameters.

The valid Process Engine values are:

OFSA_PROCESS_ENGINE

PROCESS_ ENGINE_CD	PROCESS_ENGINE_NAME	DESCRIPTION
0	Performance Analyzer Engine	Performance Analyzer Allocation ID
2	Risk Manager Engine	Risk Manager Process ID
3	Transfer Pricing Engine	Transfer Pricing Process ID
4	Balance & Control Engine	Balance & Control Data Correction Processing ID
14	Transformation Engine	Transformation ID

Processing Engine Step

OFSA multiprocessing enables you to designate a set of multiprocessing parameters to be used for a specific step within a given Processing Engine. The Processing Engine Step identifies a particular phase of an OFSA process. Processing Engine Steps are reserved names specific to each OFSA Processing Engine.

Each Processing Engine Step Name applies to a specific Processing Engine Code. The list of valid Processing Engine Steps and the Processing Engine Code for which they apply is as follows:

Process Engine CD	Step Name
0	DEFAULT
0	Page:<space>###
2	Client Data by Prod
2	Client Data by Prod, Org
2	Client Data by Prod, Currency
2	Monte Carlo client data
3	DEFAULT
4	DEFAULT
4	Bulk Statements
4	Cash Flow Edits
4	All Rules
14	DEFAULT

OFSA IDs

You can specify multiprocessing parameters at the OFSA ID level to override any parameters assigned at the Processing Engine level. This enables you to individualize your multiprocessing options to handle situations unique to specific OFSA IDs.

A list of valid OFSA IDs is obtained from the OFSA_CATALOG_OF_IDS table. Only IDs of the following types are available for processing:

- Performance Analyzer Allocation ID
- Risk Manager Process ID
- Transfer Pricing Process ID

- Balance & Control Data Correction Process ID
- Transformation ID

Defining Multiprocessing

Defining Multiprocessing is the process of associating Multiprocessing parameters to OFSA Processing IDs and Processing Engines. Included in this section are the following topics:

- Parameter Tables
- How to Specify Parameters

Parameter Tables

To define multiprocessing, you insert data into the following objects:

- OFSA_PROCESS_ID_RUN_OPTIONS
- OFSA_PROCESS_ID_STEP_RUN_OPT
- OFSA_PROCESS_ID_RUN_OPTIONS_V (Read Only View)

TABLE_NAME	DISPLAY_NAME	DESCRIPTION
OFSA_PROCESS_ID_RUN_OPTIONS	Process ID Run Options	This table specifies the OFSA Process ID for a single Process Engine.
OFSA_PROCESS_ID_STEP_RUN_OPT	Process ID Step Run Options	This table specifies the OFSA Process ID options to include specific step options.
OFSA_PROCESS_ID_RUN_OPTIONS_V	Process ID Run Options Views	A read-only view based on OFSA_PROCESS_ID_RUN_OPTIONS and OFSA_PROCESS_ID_STEP_RUN_OPT tables.

These tables are each described as follow:

OFSA_PROCESS_ID_RUN_OPTIONS

COLUMN_NAME	DISPLAY_NAME	DESCRIPTION
SYS_ID_NUM	System ID Number	Process ID System ID Number

PROCESS_ENGINE_CD	Process Engine Code	Process Engine Code that run this Process ID

OFSA_PROCESS_ID_STEP_RUN_OPT

COLUMN_NAME	DISPLAY_NAME	DESCRIPTION
SYS_ID_NUM	System ID Number	Process ID System ID Number
STEP_NAME	Step Name	The step of the Process ID getting the Process Data Slices Code and Process Partition Code
NUM_OF_PROCESSES	Number of Processes	Number of Processes
PROCESS_DATA_SLICES_CD	Process Data Slices Code	Process Data Slices Code used by this Process ID in this step
PROCESS_PARTITION_CD	Process Partition Code	Process Partition code used by this Process ID in this step

OFSA_PROCESS_ID_RUN_OPTIONS_V

Table Priority	Column Order	COLUMN_NAME	DISPLAY_NAME	DESCRIPTION
9	1	SYS_ID_NUM	System ID Number	Process ID System ID Number
9	2	PROCESS_ENGINE_CD	Process Engine Code	Process Engine Code that run this Process ID
9	3	STEP_NAME	Step Name	The step of the Process ID getting the Process Data Slices Code and Process Partition Code
9	4	NUM_OF_PROCESSES	Number of Processes	Number of Processes
9	5	PROCESS_DATA_SLICES_CD	Process Data Slices Code	Process Data Slices Code used by this Process ID in this step

How to Specify Parameters

The setup of multiprocessing is broken down into the following steps:

- Identify Assignment Level
- Assign Unit of Work
- Assign Worker Processes
- Assign Unit of Work Servicing Methodology

For each step, the relevant multiprocessing parameters are described. Some applications override the multiprocessing configuration in order to handle special processing conditions. The Application Overrides section explains the special processing conditions.

To avoid data entry errors, entering data into the OFSA multiprocessing parameter tables in the following order:

1. OFSA_PROCESS_DATA_SLICES
2. OFSA_PROCESS_DATA_SLICES_DTL
3. OFSA_PROCESS_ID_RUN_OPTIONS
4. OFSA_PROCESS_ID_STEP_RUN_OPT

> **Note:** You do not need to enter new data into OFSA_PROCESS_DATA_SLICES and OFSA_PROCESS_DATA_SLICES_DTL unless you are specifying customized Unit of Work definitions.

Identify Assignment Level

The SYS_ID_NUM and STEP_NAME columns in the OFSA_PROCESS_ID_STEP_RUN table identify the Assignment Level for multiprocessing. In the SYS_ID_NUM column you insert either a Process Engine Code or a specific SYS_ID_NUM for an individual Process ID. In the STEP_NAME column, you insert one of the valid STEP_NAME values.

The OFSA_PROCESS_ENGINE table identifies the valid values for the SYS_ID_NUM column for OFSA multiprocessing for processing engine default settings.

The OFSA_CATALOG_OF_IDS table identifies the valid values for the SYS_ID_NUM column for OFSA multiprocessing at the individual process ID level.

Because an application can have all alternatives configured, it is important to understand the order in which the application resolves the multiprocessing parameter upon application processes execution. The order is:

1. Step of a Process ID for an engine
2. Process ID for an engine
3. Step of all Process IDs for an engine
4. All Process IDs for an engine

Step of a Process ID for an Engine

At this level, a specific Step of a Process ID for the Engine is processed using the designated multiprocessing parameters.

In order to specify multiprocessing parameters for a specific Process ID, you must first insert a record into OFSA_PROCESS_ID_RUN_OPTIONS to designate the Processing Engine Code for that Process ID.

For this option set:

```
OFSA_PROCESS_ID_RUN_OPTIONS.SYS_ID_NUM =
OFSA_PROCESS_ID_RUN_OPTIONS.PROCESS_ENGINE_CD =
OFSA_PROCESS_ID_STEP_RUN_OPT.SYS_ID_NUM
```

AND

```
OFSA_PROCESS_ID_STEP_RUN_OPT.STEP_NAME='<Step Name>'
```

where <Step Name> conforms to the valid syntax specified by application. The list of valid Step Names is described in Multiprocessing Assignment Levels.

Process ID for an Engine

At this level, a specific Process ID for the Engine is processed with the designated multiprocessing parameters.

In order to specify multiprocessing parameters for a specific Process ID, you must first insert a record into OFSA_PROCESS_ID_RUN_OPTIONS to designate the Processing Engine Code for that Process ID.

For this option set:

```
OFSA_CATALOG_OF_IDS.SYS_ID_NUM =
OFSA_PROCESS_ID_RUN_OPTIONS.SYS_ID_NUM =
OFSA_PROCESS_ID_STEP_RUN_OPT.SYS_ID_NUM
```

AND

```
OFSA_PROCESS_ID_STEP_RUN_OPT.STEP_NAME='DEFAULT'
```

Step for all Process IDs for an Engine

At this level, a specific Step of the Process IDs for the Engine are processed using the designated multiprocessing parameters.

For this option set:

```
OFSA_PROCESS_ID_RUN_OPTIONS.SYS_ID_NUM =
OFSA_PROCESS_ID_RUN_OPTIONS.PROCESS_ENGINE_CD =
OFSA_PROCESS_ID_STEP_RUN_OPT.SYS_ID_NUM
```

AND

```
OFSA_PROCESS_ID_STEP_RUN_OPT.STEP_NAME='<Step Name>'
```

where <Step Name> conforms to the valid syntax specified by application. The list of valid Step Names is described in Multiprocessing Assignment Levels.

All Process IDs for an Engine

At this level, all Process IDs for the specified Engine are processed using the designated multiprocessing parameters.

Note: Risk Manager Process ID does not allow this configuration.

For this option set:

```
OFSA_PROCESS_ID_RUN_OPTIONS.SYS_ID_NUM =
OFSA_PROCESS_ID_RUN_OPTIONS.PROCESS_ENGINE_CD =
OFSA_PROCESS_ID_STEP_RUN_OPT.SYS_ID_NUM
```

AND

```
OFSA_PROCESS_ID_STEP_RUN_OPT.STEP_NAME='DEFAULT'
```

Assign Unit of Work

For the Assignment Level entered into OFSA_PROCESS_ID_STEP_RUN_OPT, enter a valid PROCESS_DATA_SLICES_CD value. Refer to Units of Work for information regarding the PROCESS_DATA_SLICES_CD values.

Assign Worker Processes

For the Assignment Level entered into OFSA_PROCESS_ID_STEP_RUN_OPT, enter an integer into the NUM_OF_PROCESSES field to specify the number of worker processes.

Assign Unit of Work Servicing

For the Assignment Level enter into OFSA_PROCESS_IT_STEP_RUN_OPT, enter either *0*, *1*, or *2* into the PROCESS_PARTITION_CD field to specify the Unit of Work Servicing methodology. Refer to Unit of Work Servicing for information about what each of these codes means. The only acceptable values for the PROCESS_ PARTITION_CD field are *0*, *1*, or *2*. No other values are allowed.

Engine Overrides

For some conditions, the OFSA Processing Engines override the multiprocessing definition for an assignment level. The overrides are as follows:

Transfer Pricing

Transfer Pricing configures the data slicing columns automatically using the Product Leaf Column defined in the active Configuration ID as the default slicing column for all runs. However, different steps in the same processing run can use different additional slicing columns. Bulk and propagation calculation steps, as well as Non-Cash Flow and Ledger Stat pricing / migration runs use the ORG_UNIT_ID column as an additional slicing column. The Cash Flow Transfer Pricing step also uses the ORG_UNIT_ID column if not combined with Option Cost Calculations. For Option Cost Calculation (including combined Transfer Pricing / Option Cost Calculation), the engine employs ORIGINATION_DATE as the primary slicing column with the Product Leaf Column as secondary.

Transformation ID

To prevent aggregation errors resulting from data slices that may conflict with the dimension filtering feature, the Transformation ID ignores the OFSA_PROCESS_ ID_STEP_RUN_OPT.PROCESS_DATA_SLICES_CD. The Transformation ID automatically configures the data slicing columns, ORG_UNIT_ID and COMMON_

COA_ID. If either column is excluded in the dimension filter, then it is not used as a data slicing column.

To prevent aggregation errors resulting from table partitions that may conflict with the dimension filtering feature, the Transformation ID ignores OFSA_PROCESS_ID_STEP_RUN_OPT.PROCESS_PARTITION_CD.

Risk Manager

Ignores OFSA_PROCESS_ID_STEP_RUN_OPT.PROCESS_PARTITION_CD because of the need to order units of work specifically. (that is, the same leaves can exist in more than one table and / or table partition).

Risk Manager configures the data slicing columns automatically using the Product Leaf Column defined in the active Configuration ID as the default slicing column for all runs. The Risk Manager engine adds additional slicing column as follows based upon the parameters specified in the Risk Manager Process ID:

- If Product / Organizational Unit functional dimensions are selected, the engine adds ORG_UNIT_ID as an additional slicing column.
- If Product / Currency functional dimensions are selected, the engine adds ISO_CURRENCY_CD as an additional slicing column.
- If Product / Organizational / Currency functional dimensions are selected, the engine adds ORG_UNIT_ID and ISO_CURRENCY_CD as additional slicing columns.

Columns specified in PROCESS_DATA_SLICES_DTL that are already implicitly added by the RM engine (that is, ORG_UNIT_ID if Product / Organizational Unit functional dimensions are selected) are ignored. All other columns specified in this tables are added to the slicing column list.

Performance Analyzer

Ignores OFSA_PROCESS_ID_STEP_RUN_OPT.PROCESS_PARTITION_CD for additive set operation because of the need to order units of work specifically.

Tuning Multiprocessing

Tuning for optimal multiprocessing settings is an exercise similar to tuning a database. It involves experimentation with different settings under different load conditions

.Database Bound versus Engine Bound Jobs

OFSA jobs fall into the following two categories:

- Database bound—Those jobs that spend more time within database manipulations
- Engine bound—Those jobs whose calculations are complex, thus the time spent with database operations is small compared to the amount of time doing calculations.

The following table lists OFSA jobs by Processing Engine and identifies whether the job is usually database bound or Engine bound.

Application	Job Type	Generic Job Type	OFSA/DB Bound	MP Enabled	Comments
Balance & Control	Cash Flow Edits	Row by Row	DB or OFSA	Yes	If a large number of corrections need to be done, this may become OFSA bound
Balance & Control	Row by Row	Row by Row	DB or OFSA	Yes	As the number of rules in the correction processing ID increase, the process may become OFSA bound
Balance & Control	Bulk	Bulk	DB	Yes	
Performance Analyzer	Straight Ledger	Row by Row	DB	Yes	No percent or table ID. Ledger_Stat reading is MP enabled, Ledger_Stat writing is not.
Performance Analyzer	Percent Distribution	Row by Row	DB	Yes	
Performance Analyzer	Table ID	Row by Row	DB	Yes	
Performance Analyzer	Lookup Table ID	Bulk	DB	Yes	
Performance Analyzer	Detail	RBR/Bulk	DB	Yes	Row by Row for Ledger_Stat allocation, update to detail table is bulk
Transfer Pricing	Rate Migration	Bulk	DB	Yes	
Transfer Pricing	Bulk Transfer Pricing	Bulk	DB	Yes	
Transfer Pricing	Non-Cash Flow Transfer Pricing	Row by Row	DB	Yes	
Transfer Pricing	Cash Flow Transfer Pricing	Row by Row	OFSA	Yes	

Application	Job Type	Generic Job Type	OFSA/DB Bound	MP Enabled	Comments
Transfer Pricing	Ledger_Stat Migration	Row by Row	DB	Yes	
Risk Manager	Detail Processing (Current position, Gap, Market Value)	Row by Row	OFSA	Yes	All processing except Formula Leaves and Auto Balancing
Risk Manager	Formula Leaves	Row by Row	OFSA	No	
Risk Manager	Auto Balancing	Row by Row	OFSA	No	
Transformation	Ledger	Row by Row	DB	Yes	Dimension Filtering (aggregation) and complex OFSA filters affect performance
Transformation	Risk Manager	Row by row	DB	Yes	Dimension Filtering (aggregation) affects performance
Transformation	Roll up	Other	DB	No	

The scalability of database bound jobs is largely determined by size of the database server. The scalability of Engine bound jobs is determined by the size of the application server.

Tuning the OFSA Database from the Application Layer

Despite the many multiprocessing options, tuning the OFSA database from the application layer is achieved by following a simple process. The process is as follows:

1. Identify the OFSA job types that are used by your organization.
2. For each job type, time the runs for a series of OFSA_PROCESS_ID_STEP_RUN_OPT.NUM_OF_PROCESSES settings.
3. Based on the results, determine the appropriate setting per application.

In general, it is recommended that for each process type, start with a OFSA_PROCESS_ID_STEP_RUN_OPT.NUM_OF_PROCESSES setting of 1, and then double the setting until it is equal to the number of processors available on the application server for application bound jobs, and equal to the number of processors available on the database server for database bound jobs.

Generally, as the OFSA_PROCESS_ID_STEP_RUN_OPT.NUM_OF_PROCESSES setting is increased, you can expect performance improvements up to a certain point. After that, processing times level off and then start increasing. Testing has shown that for all applications, the point at which processing time starts to increase occurs after the OFSA_PROCESS_ID_STEP_RUN_OPT.NUM_OF_PROCESSES setting has exceeded the number of processors on the machine. Also, as the NumProcesses setting approaches the number of processors, the performance improvements are minimal.

Note: You may need to change the OFSA_PROCESS_ID_STEP_RUN_OPT.NUM_OF_PROCESSES settings for applications at different stages in the production cycle to achieve optimal performance.

Special considerations need to be made when multiple OFSA jobs are run at the same time. Different types of jobs use resources differently. You may want to look into a scheduling tool to help you schedule dissimilar jobs to run at different times.

Ledger_Stat Updating

Transfer Pricing, Ledger_Stat migration and Performance Analyzer update the LEDGER_STAT table using an update/insert methodology where an update is attempted and, if no rows are affected, an insert is performed. This methodology prevents OFSA from performing Ledger_Stat updates in parallel. The result is that when Ledger_Stat is updated (either because the Ledger_Stat buffer has filled or the process has ended) the updating is done by only one process. All other processes must wait for the updating to be completed.The result is that as the ratio between rows written to Ledger_Stat and rows read to Ledger_Stat increases, the time spent writing Ledger_Stat dominates the time spent reading, resulting in drastically reduced scalability. Testing indicates that many Performance Analyzer allocations fall into this category.

Special Considerations

Because of the nature of parallel processing performed by OFSA, different processes tend to need to access the same tables at the same time. Unless care is taken in designing the layout of the database tables, this can lead to I/O contention, which in turn, can reduce scalability.

Migration from OFSA 3.5/4.0

Multiprocessing settings are not preserved when upgrading from a previous release of OFSA to FDM 4.5. The FDM 4.5 Database Upgrade Process reverts all multiprocessing settings to default values. Because the default values for multiprocessing in Release 4.5 are not the same as the default multiprocessing settings in OFSA 3.5/4.0, Oracle recommends that you review this section after you complete your 4.5 upgrade to identify potential performance issues.

Upgrading from OFSA 3.5/4.0 Default Multiprocessing

This section describes how to implement multiprocessing in Release 4.5 to replicate the default settings of OFSA 3.5/4.0. If you did not change the default Org/Product Leaf Partitioning or Thread Division multiprocessing options in OFSA 3.5/4.0, then follow the instructions in this section to replicate these settings in your FDM 4.5 database.

The default WHERE conditions for SQL generated by the OFSA processing engines in Release 4.5 is different than the default generated by the OFSA 3.5/4.0 multiprocessing implementation. The 4.5 default Unit of Work definitions, which determine what columns are used in the WHERE conditions, are not the same as the default Unit of Work definitions in OFSA 3.5/4.0. This is true even if you did not customize your multiprocessing in OFSA 3.5/4.0.

By updating the OFSA_PROCESS_ID_STEP_RUN_OPT table to appear, you set the 4.5 multiprocessing options to replicate the OFSA 3.5/4.0 default settings.

SYS_ID_NUM	STEP_NAME	NUM_OF_PROCESSES	PROCESS_DATA_SLICE_CD	PROCESS_PARTITION_CD
0	DEFAULT	??	1	0
2	Client Data by Prod	??	1	0
2	Client Data by Prod, Currency	??	1	0
2	Client Data by Prod, Org	??	1	0
2	Monte Carlo Client Data	??	1	0
3	DEFAULT	??	1	0
4	DEFAULT	??	1	0
14	DEFAULT	??	1	0

Set the NUM_OF_PROCESSES value to equal the NumProcesses parameter that you used in OFSA 3.5/4.0. This is the Number of Worker Processes employed for the multiprocessing operations. In OFSA 3.5/4.0, this parameter is found in the [parallel] section of each application-specific .INI file on the server.

By setting the columns in the OFSA_PROCESS_ID_STEP_RUN_OPT table to these values, the OFSA 4.5 Knowledge Engines processes using the same multiprocessing parameters as the default settings from OFSA 3.5/4.0.

Upgrading from OFSA 3.5/4.0 Customized Multiprocessing

This section describes how to implement in Release 4.5 any customized Unit of Work definitions or Thread Divisions settings from OFSA 3.5/4.0.

In order to maintain equivalent performance for Release 4.5 in a customized multiprocessing environment, you must either re-tune based upon the new (default) SQL generated from Release 4.5, or adjust the 4.5 multiprocessing options to replicate the behavior for your OFSA 3.5/4.0 implementation. Because the new multiprocessing parameter format for Release 4.5 does not directly correspond to the way that multiprocessing was specified in OFSA 3.5/4.0, it may be simpler to re-tune your database to accommodate any OFSA Engine SQL changes caused by the new default 4.5 multiprocessing parameters. However, if you convert your previous multiprocessing settings into FDM 4.5 instead of re-tuning, the following section provides information to assist you in this conversion. This section describes how multiprocessing in OFSA 3.5/4.0 works in comparison to Release 4.5. There are 3 different options to be examined:

- Units of Work
- Unit of Work Servicing
- Worker Processes

Note: Regardless of the approach you select, Oracle recommends that you reserve sufficient time to address database and application tuning for 4.5 after completing your upgrade. The issues involved require some investigation and experimentation in order to achieve the desired performance.

Units of Work

The first step in re-implementing multiprocessing settings from an OFSA 3.5/4.0 database into FDM 4.5 is to identify any Unit of Work definitions that are not

seeded in FDM 4.5. Once you have identified all of the Unit of Work definitions used in your OFSA 3.5/4.0 implementation (both seeded and custom), you need to assign the appropriate definitions to your processes in the FDM 4.5 database.

Identifying Custom Unit of Work Definitions

OFSA 3.5/4.0 allowed up to two columns to be used for the Unit of Work definition. These columns were specified in either the database (the TSER_APP_PROCESS table) or in the application-specific INI file.

When specified in the database, the Unit of Work columns were defined by setting the PARTITION_CD column to *1* (*On*) or *0* (*Off*) for the ORG Column Type and the PROD Column Type in TSER_APP_PROCESS. The ORG Column Type was defined as the ORG_UNIT_ID column. The PROD Column Type was determined based upon the specific OFS application:

Application	Product Leaf Column
Balance & Control - Data Correction Processing	COMMON_COA_ID
Performance Analyzer - Allocation ID	COMMON_COA_ID
Risk Manager - Process ID	Product Leaf in active Configuration ID
Transfer Pricing - Process ID	Product Leaf in active Configuration ID
Transformation Engine	COMMON_COA_ID

When specified in the application-specific .INI file, the columns were identified by setting specific parameters = 0 (Off) or 1 (On). These parameters were:

Leaf	Setting	Description
RowOrgLeaf	0	Use org leaf for calculating units of work for Row by Row processing.
	1	Use org leaf for calculating units of work for Row by Row processing
RowProdLeaf	0	Do not use product leaf for calculating units of work for Row by Row processing
	1	Use product leaf for calculating units of work for Row by Row processing

Leaf	Setting	Description
BulkOrgLeaf	0	Do not use org leaf for calculating units of work for Bulk processing
	1	Use org leaf for calculating units of work for Bulk processing
BulkProdLeaf	0	Do not use Prod leaf for calculating units of work for Bulk processing
	1	Use Prod leaf for calculating units of work for Bulk processing

Again, the Org Leaf is the ORG_UNIT_ID column and the Prod Leaf is determined based upon the application.

In order to identify the custom Unit of Work settings from OFSA 3.5/4.0, analyze the assignments in TSER_APP_PROCESS or the application-specific INI file for each application. Any partitioned columns in OFSA 3.5/4.0 for which a PROCESS_DATA_SLICES_CD value is not already provided in FDM 4.5 require a customized Unit of Work definition.

For example, assume the following data in TSER_APP_PROCESS:

Example 1

TSER_PRODUCT	PROCESS_TYPE	COLUMN_TYPE	PARTITION_CD
0	RBR	ORG	1
0	RBR	PROD	1
0	BULK	ORG	1
0	BULK	PROD	1

The assignments mean that the user is processing the Allocation ID engine using ORG_UNIT_ID and COMMON_COA_ID as the partition definition for all processes, both Row by Row and Bulk (this is actually the default for OFSA 3.5/4.0). The TSER_PRODUCT values from OFSA 3.5/4.0 are the same as the PROCESS_ENGINE_CD values of Release 4.5. In this case, TSER_PRODUCT = 0 indicates Performance Analyzer.

Unit of Work definitions for FDM 4.5 are specified in the OFSA_PROCESS_DATA_SLICES and OFSA_PROCESS_DATA_SLICES_DTL tables. For every unique Unit of Work combination used in OFSA 3.5/4.0, you need to populate these tables appropriately.

For Example 1, where the user is processing the Allocation engine with the default Unit of Work definition (ORG_UNIT_ID, COMMON_COA_ID), there is no need to create a new PROCESS_DATA_SLICES_CD value. As you can see by examining the OFSA_PROCESS_DATA_SLICES_DTL table, FDM 4.5 seeds a PROCESS_DATA_SLICES_CD with (ORG_UNIT_ID, COMMON_COA_ID) as the columns used for the Unit of Work definition

However, because the default PROCESS_DATA_SLICES_CD assignment for Performance Analyzer is *3* (COMMON_COA_ID), you need to update the OFSA_PROCESS_ID_STEP_RUN_OPT table with the (ORG_UNIT_ID, COMMON_COA_ID) PROCESS_DATA_SLICES_CD value See Assigning Unit of Work Definitions for details on how to perform this assignment.

Following is another example:

Example 2

TSER_PRODUCT	PROCESS_TYPE	COLUMN_TYPE	PARTITION_CD
2	RBR	ORG	1
2	RBR	PROD	1
2	BULK	ORG	1
2	BULK	PROD	1

In this example, the user is processing Risk Manager with the default Unit of Work definition. However, the Product Leaf Column is not always COMMON_COA_ID. Rather, it is determined based upon the active Configuration ID. For this example, assume that the active Configuration ID is using RM_COA_ID as the Product Leaf. The Unit of Work is therefore (ORG_UNIT_ID, RM_COA_ID). Since FDM 4.5 does not seed such a definition into OFSA_PROCESS_DATA_SLICES_DTL, we need to create a customized Unit of Work definition for FDM 4.5.

To do this, insert data into the OFSA_PROCESS_DATA_SLICES and OFSA_PROCESS_DATA_SLICES_DTL tables as follows:

OFSA_PROCESS_DATA_SLICES

PROCESS_DATA_SLICES_CD
4

OFSA_PROCESS_DATA_SLICES_DTL

PROCESS_DATA_SLICES_CD	PROCESS_DATA_SLICES_SEQ	COLUMN_NAME
4	1	ORG_UNIT_ID
4	2	RM_COA_ID

Assigning Unit of Work Definitions

Once you have identified (and if necessary created) the Unit of Work definitions used in your OFSA 3.5/4.0 implementation, you must assign them to the appropriate processes and processing engines. In OFSA 3.5/4.0, Unit of Work definitions could be assigned only at the engine level for two categories of processing: Row by Row and Bulk. FDM 4.5 provides much more flexibility, enabling you to specify Unit of Work definitions at the engine level, engine step level, or individual ID level. However, none of these levels corresponds directly with the Row by Row and Bulk options available in OFSA 3.5/4.0.

The strategy for converting the OFSA 3.5/4.0 multiprocessing settings to FDM 4.5 varies depending upon the application:

Non-Risk Manager Processes

The same strategy applies for the following applications:

- Balance & Control
- Performance Analyzer
- Transfer Pricing
- Transformation Engine

If you used the same Unit of Work Definition for both Bulk and Row by Row processing in OFSA 3.5/4.0 (on an application by application basis), then assign that definition to the DEFAULT Step Name in OFSA_PROCESS_ID_STEP_RUN_OPT for SYS_ID_NUM=0 (Performance Analyzer), 3 (Transfer Pricing), 4 (Balance & Control) or 14 (Transformation Engine). This sets all IDs of the specified application to run using that assigned Unit of Work definition.

If you did NOT use the same Unit of Work Definition for both Bulk and Row by Row processing, then assign the most commonly used Unit of Work definition to the DEFAULT Step Name for appropriate SYS_ID_NUM. For example, if the majority of your Allocation IDs are Bulk processing, then assign that Unit of Work definition as the DEFAULT for SYS_ID_NUM=0. Then, for each of your Row by

Row Allocations, create a customized assignment in OFSA_PROCESS_ID_STEP_RUN_OPT (for the specific SYS_ID_NUM values) assigning the Row by Row Unit of Work definition to those IDs.

Risk Manager Processes

For Risk Manager, only Row by Row processing applies. Therefore, assign the OFSA 3.5/4.0 Row by Row Unit of Work definition to the following Step Names in OFSA_PROCESS_ID_STEP_RUN_OPT for SYS_ID_NUM=2 (Risk Manager):

- Client Data by Prod
- Client Data by Prod, Currency
- Monte Carlo Client Data
- Client Data by Prod, Org

This sets all Risk Manager Process IDs to run using that assigned Unit of Work definition.

Unit of Work Servicing

Unit of Work Servicing in Release 4.5 is similar in concept to the Thread Division concept of OFSA 3.5/4.0, although it is not exactly the same. The Unit of Work Servicing feature in 4.5 works directly with distinct table partitions, rather than only with distinct tables. However, despite this difference, the Thread Division settings from OFSA 3.5/4.0 are equivalent to the new Unit of Work Servicing settings for Release 4.5.

Thread Division in OFSA 3.5/4.0 was specified in the application-specific INI files on the server. The default Thread Division for OFSA 3.5/4.0 was Normal (ThreadDivision = 0), which is equivalent to the Single Servicing Unit of Work Servicing methodology for Release 4.5. If you did not deviate from the default Thread Division in OFSA 3.5/4.0, you do not need to perform any conversion to 4.5 for this setting, since Single Servicing is the default Unit of Work Servicing methodology for Release 4.5.

However, if you did deviate from the default Thread Division setting for any of the applications, then you need to specify the equivalent Unit of Work Servicing value in the PROCESS_PARTITION_CD column of OFSA_PROCESS_ID_STEP_RUN_

OPT. The Unit of Work Servicing codes are the same values as the Thread Division codes:

Thread Division Code		Unit of Work Servicing
0 ('Normal')	is equivalent to	0 ('Single Servicing')
1 ('Preferred')	is equivalent to	1 ('Cooperative Servicing')
2 ('Required')	is equivalent to	2 ('Dedicated Servicing')

Since Thread Division codes were assigned at an application basis, update the PROCESS_PARTITION_CD column for all processes of a given application with the desired Unit of Work Servicing. For example, if the Thread Division for 3.5/4.0 Performance Analyzer was set to 1 - Preferred, then set the PROCESS_PARTITION_CD=1 in OFSA_PROCESS_ID_SET_RUN_OPT for all Performance Analyzer entries, which includes SYS_ID_NUM=0 (Step Name=DEFAULT) as well as any other entries for specific Allocation SYS_ID_NUM values or Steps.

Worker Processes

The Worker Processes concept in Release 4.5 is equivalent to the NumProcesses setting in the 3.5/4.0 application-specific .INI files. Since the NumProcesses setting was assigned at an application basis in 3.5/4.0, update the NUM_OF_PROCESSES column for all processes of a given application with the desired Worker Processes value. For example, if the NumProcesses for 3.5/4.0 Risk Manager was set to 4, then set the NUM_OF_PROCESSES = 4 in OFSA_PROCESS_ID_SET_RUN_OPT for all Risk Manager entries. These include SYS_ID_NUM=2 for all Steps, as well as any other entries for specific Risk Manager SYS_ID_NUM values or Steps that are also in the OFSA_PROCESS_ID_SET_RUN_OPT table.

Examples

The following tables illustrate examples of valid multiprocessing parameters:

Parameters for OFSA_PROCESS_ID_STEP_RUN_OPT

SYS_ID_NUM	STEP_NAME	NUM_OF_PROCESSES	PROCESS_DATA_SLICES_CD	PROCESS_PARTITION_CD
0	DEFAULT	1 to 100	1 to 100	0, 1, or 2

SYS_ID_NUM	STEP_NAME	NUM_OF_ PROCESSES	PROCESS_ DATA_SLICES_ CD	PROCESS_ PARTITION_CD
###123	DEFAULT	1 to 100	1 to 100	0, 1, or 2
###123	Page:<space>###	1 to 100	1 to 100	0, 1, or 2
0	Page:<space>###	1 to 100	1 to 100	0, 1, or 2
2	Client Data by Prod	1 to 100	1 to 100	0, 1, or 2
2	"Client Data by Prod, Org "	1 to 100	1 to 100	0, 1, or 2
2	"Client Data by Prod, Currency"	1 to 100	1 to 100	0, 1, or 2
2	Monte Carlo client data	1 to 100	1 to 100	0, 1, or 2
###456	Client Data by Prod	1 to 100	1 to 100	0, 1, or 2
###456	"Client Data by Prod, Org "	1 to 100	1 to 100	0, 1, or 2
###456	"Client Data by Prod, Currency"	1 to 100	1 to 100	0, 1, or 2
###456	Monte Carlo client data	1 to 100	1 to 100	0, 1, or 2
3	DEFAULT	1 to 100	1 to 100	0, 1, or 2
###789	DEFAULT	1 to 100	1 to 100	0, 1, or 2
4	DEFAULT	1 to 100	1 to 100	0, 1, or 2
4	Bulk Statements	1 to 100	1 to 100	0, 1, or 2
4	Cash Flow Edits	1 to 100	1 to 100	0, 1, or 2
4	All Rules	1 to 100	1 to 100	0, 1, or 2
###321	DEFAULT	1 to 100	1 to 100	0, 1, or 2
###321	Bulk Statements	1 to 100	1 to 100	0, 1, or 2
###321	Cash Flow Edits	1 to 100	1 to 100	0, 1, or 2
###321	All Rules	1 to 100	1 to 100	0, 1, or 2
14	DEFAULT	1 to 100	1 to 100	0, 1, or 2

SYS_ID_NUM	STEP_NAME	NUM_OF_ PROCESSES	PROCESS_ DATA_SLICES_ CD	PROCESS_ PARTITION_CD
###654	DEFAULT	1 to 100	1 to 100	0, 1, or 2

20

Request Queue

This chapter provides information on both single-host and multi-host Request Queue.

Single-Host Request Queue

Request Queue links a remote client application with the UNIX-based server to spawn knowledge engines on the server. This frees the client for other tasks while the knowledge engine is performing its task.

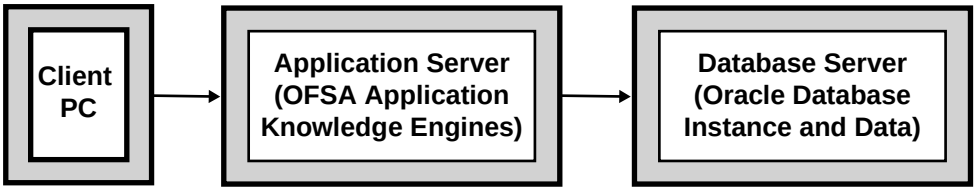
Without Request Queue in place remote clients cannot spawn knowledge engines on the server.

Request Queue also maintains a log file to track knowledge engine requests, provides status updates for each request and returns error messages if a problem arises in the spawning or processing of a request.

Request Queue interacts with three hardware components. These include the remote client, the Oracle Financial Services (OFS) application server and the database server. The knowledge engine is run on the application server; the database instance is located on the database server.

The following diagram illustrates the relationship between the remote client, the application server and the database server.

This illustration separates the application and database servers into two entities, however both the application and the database instance can reside on the same server.



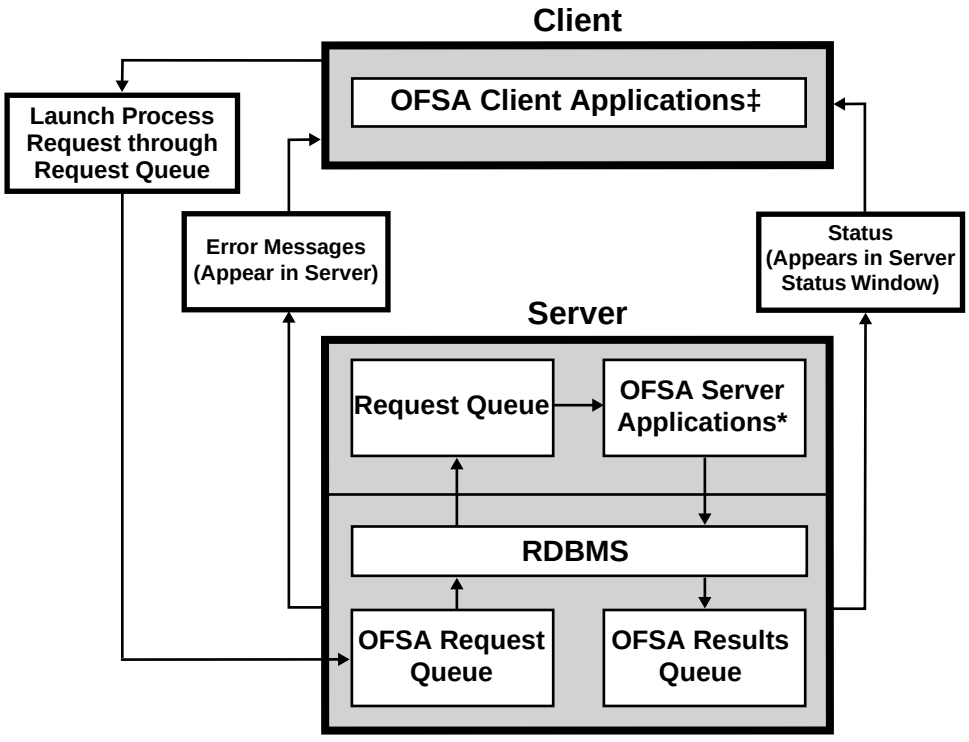
Using Request Queue

You can run only one instance of Request Queue against a database.

Spawning a knowledge engine on the server follows a three-step process, as follows:

1. The user places a request, through the remote client application, to the OFSA_REQUEST_QUEUE table, which is located in the Oracle Financial Services Applications (OFSA) database.
2. Request Queue retrieves the request from the table and spawns the knowledge engine.
3. Request Queue updates the exit code and status of the application after the process is completed.

The function of Request Queue is shown in the following diagram.



** Included under OFSA server applications are the following: Balance and Control, Performance Analyzer, Risk Manager, Transfer Pricing, Transformation and RQ Test*

‡ Included under OFSA client applications are the following: Balance and Control, Performance Analyzer, Risk Manager and Transfer Pricing

Launching Request Queue

Request Queue is not automatically activated when the OFS applications are installed on the server and client. Activating Request Queue is a manual process, performed by a System Administrator or DBA, using either the **rq** command, which is a UNIX shell script designed to set the required environment variables automatically, or by launching Request Queue directly.

> **Note:** Oracle recommends that the person responsible for launching Request Queue is also responsible for maintaining the applications.

If you decide to launch Request Queue directly you need to set up the environment variables manually, according to the requirements of the server vendor. Use the rq script as a reference for the variables you need to address. See Chapter 6, "UNIX Server Installation and Configuration" for detailed information on setting up the server environment variables specific to the server on which you are installing the OFSA server-centric software.

Using the rq Script to Set Up Request Queue

The command line interface of Request Queue is used to set the operational parameters of the application and, when necessary, to kill the Request Queue currently running against the database.

Once the operational parameters have been set they remain in place as long as the current Request Queue application is active. If you want to change these parameters you need to kill the current Request Queue application, reset the operational parameters using the command line interface and then launch Request Queue again.

All of the command line switches are listed in the following two tables.

Setting Operational Parameters

This table includes switches used to set the operational parameters of Request Queue.

Command Switches	Description	Mandatory/Optional
-b	Runs Request Queue as a background process. Request Queue runs in the foreground if **-b** is omitted from the command line. Running Request Queue in the foreground locks the terminal from which Request Queue is launched until this instance of Request Queue is killed.	Optional
-m num_blocks	This switch sets the maximum size of the logfile (in 1024 byte blocks). The default setting is 512. The logfile flushes lines of text at the beginning of the file when the limit is reached. Setting this value to zero results in unlimited logfile size.	Optional
-c num_blocks	Sets the amount of text (in 1024 byte blocks) retained in the logfile after the maximum size has been reached and text is flushed from the beginning of the file. This setting specifies the amount of text retained in the logfile, not the amount of text to be flushed. The default setting is 256. If this switch is not set, the text retained in the logfile equals 256.	Optional

Command Switches	Description	Mandatory/Optional
-i interval	Sets the time interval (in seconds) for Request Queue to poll the Request Queue table. The default setting is 3 seconds. The polling interval affects the number of seconds a request sits in the Request Queue table before it is initiated. Sometimes the initiation of a request does not occur within the time interval. Generally, the cause of this is a greater number of requests in the table than Request Queue can initiate within the polling interval. This setting also works with the ServerTimeOut parameter in the client ofs.ini file to affect Request Queue behavior on the client. Refer to Chapter 7, "Client Software Installation and Configuration" for information about how the -i parameter affects the ServerTimeOut setting.	Optional
database_alias	This is the name (alias) of the database that Request Queue polls. If the specified alias is referenced in the ofs.ini file, the database connectivity parameters from that file are used. Otherwise, the parameters in the tnsnames file are used.	Mandatory
logfile	Sets up the log file that Request Queue uses to log application and process information as well as error messages for each spawned knowledge engine. Refer to the information described in the -l switch, as it affects the logfile output. All information, including error messages, for a spawned knowledge engine runs in the background and will be lost if a log file is not specified. For a spawned knowledge engine run in the foreground only, the information returned to the window is available to the user. This status information is continually flushed as additional status information is returned to the window.	Optional

Command Switches	Description	Mandatory/Optional
`-l {remove|keep|none}`	remove: Remove output files after successful completion of process. If process does not complete successfully (a non-zero return status) the file is not removed. Remove is the default behavior. keep: Keep files. Do not delete them on successful completion. none: Do not use separate output files, log all output into the rq log.	

Killing a Previously Launched Request Queue Instance

This table includes switches used to kill a previously launched Request Queue application.

Command Switches	Description	Optional/Mandatory
-n	Instructs a newly launched Request Queue to exit if it encounters a previously launched Request Queue running against the target database.	Optional
-k	Kills a previously launched Request Queue after all spawned child processes are completed.	Optional
-K	Kills a previously launched Request Queue as well as all spawned child processes.	Optional
-w seconds	Specifies for Request Queue to wait for the existing (running) Request Queue to exit. The optional *seconds* parameter specifies the maximum amount of time to wait. If the (running) Request Queue has not exited by the specified time, the newly launched Request Queue terminates.	Optional

Killing a Request Queue process does not necessarily result in the process immediately terminating. The process does not exit until all child processes that it is monitoring have ended. This is true even when the **-K** option is used. In rare situations, some child processes may not respond properly to the message to terminate and the child processes need to be killed manually using the **kill -9 <pid>** UNIX command (where **<pid>** is the process ID of the child process).

If a Request Queue refuses to terminate, look for child processes that are blocking the termination and kill them with the **kill -9** command. Once all child processes have terminated, Request Queue exits. If you kill a Request Queue instance with the **kill -9** command, process statuses in the OFSA_REQUEST_QUEUE are not updated upon process completion.

Note: Note that once a Request Queue instance is killed a new Request Queue needs to be launched from the command line.

OFS.INI Settings

There are a few settings in the OFS.INI file that control the behavior of Request Queue.

[OFSRQ]

WorkingDirectory	Sets the location for all log files for Request Queue and the applications
MaxFileSize	See **-m** in the "Setting Operational Parameters"section of this chapter.
IdealFileSize	See **-c** in the "Setting Operational Parameters" section of this chapter.

Command Line Examples

The following five examples show how the command line switches are used to establish operational parameters or kill a previously launched Request Queue instance.

The five examples include:

- Launching Request Queue in the foreground without a log file
- Launching Request Queue in the background with a log file.
- Launching Request Queue in the background with a log file, including specifications for the maximum size of the file and the amount of logged-in information to be retained after flushing.
- Launching Request Queue in the foreground without a log file but with a previously launched Request Queue, that was not anticipated, running against the targeted database.
- Killing a previously launched Request Queue instance using kill and wait switches and then launching a new Request Queue instance.

Example 1

Launching Request Queue in the Foreground Without a Log File

1. Choose an alias for the database that this instance of Request Queue will poll. (In this example the database alias is *Production*).

2. Type **`rq Production`** in the command line and press Enter.
3. Type your User ID in the login prompt and press Enter.
4. Type your password in the login prompt and press Enter.

Example 2

Launching Request Queue in the Background With a Log File

1. Choose an alias for the database that this instance of Request Queue will poll. (In this example the database alias is *Production*).
2. Type **`rq -b Production prod.log`** in the command line and press.

   Choosing the **`-b`** switch instructs Request Queue to run in the background. The name selected for the log file is prod.log.

   The following message appears if Request Queue cannot create the log file:

   ```
   ofsrq: can't open file logfile, errno = a_errno
   ```

   The *a_errno* message is returned as a number that corresponds to the UNIX system error number for the reason the file creation failed. You have to correct the problem before you can proceed. After making the correction, proceed to the next step.
3. Type your User ID in the login prompt and press Enter.
4. Type your password in the login prompt and press Enter**.**

Example 3

Launching Request Queue in the background with a log file and specifications for the maximum size of the file and the amount of logged-in information to be retained after flushing.

1. Choose an alias for the database that this instance of Request Queue will poll. (In this example the database alias is *Production*).
2. Type **`rq -b -m 512 -c 256 Production prod.log`** in the command line and press Enter.

   The **`-b`** command instructs Request Queue to run in the background.

The **-m** command, with a parameter of 512, sets the maximum size of the log file to 512kb (1024 byte blocks). When the log file grows to 512kb, it is flushed to the ideal size (**-c** switch).

The **-c** command, with a parameter of 256, sets the retained logged-in processes, after flushing, to 256kb (1024 byte blocks).

The **prod.log** command is the name assigned to the log file.

3. Type your User ID in the login prompt and press Enter.
4. Type your password in the login prompt and press Enter.

Example 4

Launching Request Queue in the foreground without a log file but with a previously launched Request Queue, that was not anticipated, running against the targeted database.

1. Choose an alias for the database that this instance of Request Queue will poll. (In this example the database alias is *Production*).
2. Type **rq Production** in the command line and press Enter.

   Because only one Request Queue instance can be running against a database the following message appears:

```
OFSA  Version X.XX.XXX
Request Queue Release X.XX
Copyright p Oracle Corporation 1995-1998
ofsrq is already running as pid 10854 on database Db1, what shall I do?
(K)ill other process and children
(k)ill other process
(A)bort this process
```

   The number for the OFSA release installed on your system and the Request Queue release appears in place of the “x”s. In a typical set up the UNIX Process ID (pid) number is assigned by the operating system and, for Request Queue, can be any number between 101 and 29999. In the example, the operating system assigned a pid of 10854.

   Database Db 1 matches the alias Production in the ofs.ini file.

3. Select one of the following three switches provided:

K (uppercase)	This terminates Request Queue 10854 immediately, including all child processes spawned by this Request Queue instance that are currently running.
k (lowercase)	This kills Request Queue 10854 after the spawned child processes currently running have been completed.
A (uppercase)	This aborts the launch of the new Request Queue instance.

4. If you abort the launching of the new Request Queue instance the current instance continues operating without interruption.
5. If you kill the previously launched Request Queue instance you need to return to the command line to launch a new Request Queue instance.
6. Type **`rq Production`** in the command line and press Enter.
7. Type your User ID in the login prompt and press Enter.
8. Type your password in the login prompt and press Enter.

Example 5

Killing a previously launched Request Queue instance using kill and wait switches and then launching a new Request Queue instance

1. Choose an alias for the database that this instance of Request Queue Request Queue will poll. (In this example the database alias is *Production*).
2. Type **`rq -K -w 30 Production`** in the command line and press Enter

   This example uses the **`-K`** command, which terminates the current Request Queue instance and all spawned child processes associated with that Request Queue instance.

The `-w` command sets a parameter of 30, indicating that the new Request Queue process will wait 30 seconds for the previous Request Queue and all of its children to terminate. The Request Queue process returns when the polling RQ terminates or when 30 seconds have elapsed.

Caution: The new Request Queue waits indefinitely for the previous Request Queue and its children to terminate if no parameter is supplied for the `-w` command.

The OFSA_REQUEST_QUEUE Table

The OFSA_REQUEST_QUEUE table contains information identifying the process(es) to be run on the server. This table is located in the OFSA database. Column names and descriptions for OFSA_REQUEST_QUEUE are provided in the following table.

Column Name	Description
Job_Num	The unique job identification number. This number is used to identify the row for updates and queries.
Login_Name	The user name of the individual requesting a server-based processing run or application execution.
Status	The status of the UNIX process after completion.
Process_ID	The process ID for the application - assigned by the UNIX operating system.
Return_Code	The exit or return code of the UNIX process.
Priority	Currently not used.
Application	An application identification number used for application-level security.
Service_Request	A label identifying the exact request. This label is used to look up the location and name of the application to be launched. The path associated with each label is defined in the ofs.ini file.
Request_Date	The timestamp the job was submitted.
Schedule_Date	Currently not used.
App_Arguments	An encrypted field containing the arguments that are passed to the application.
Host_Name	If you are running multi-host Request Queue a non-default value appears in this column.

Column Name	Description
End_Date	The timestamp the job ends.
End_Time	The time the job ends.
Language	NLS Language.
Territory	NLS Territory

Server Application Arguments

Any server-side application can receive its program arguments from either Request Queue or from prompts that are displayed if the application is attached to a tty. Any application executed by Request Queue will be a background process and will not be attached to a tty.

This section discusses the method for starting an OFSA server application when it is attached to a tty. However, the order of arguments are the same whether or not the process is executed by Request Queue.

Each application requires both common and application-specific arguments. The common arguments identify the database to which the application attaches and the user account to be used. The application-specific arguments add additional information needed to run each of the applications.

The following environment variables need to be set before the common and application-specific arguments can be set.

- OFSA_INSTALL is the OFSA installation directory
- INIPATH is OFSA_INSTALL/etc

- SHLIB_PATH or LD_LIBRARY_PATH is OFSA_INSTALL/lib
- OFSA_INSTALL/bin must be in the path environment

> **Note:** In this chapter, OFSA_INSTALL is the convention used to indicate where the OFSA software is installed in your directory structure.

Setting the Common Arguments

The process for setting the common arguments is described, as follows.

Starting the Application

Type the appropriate command at the command prompt to start the server application. The table lists the application and corresponding command.

Oracle Application	Command
Balance & Control	ofsbc
Performance Analyzer	ofspa
Risk Manager	ofsrm
Transfer Pricing	ofstp
Transformation Engine	ofste

Common Database Attachment Method

Every server-side application has to be attached to the database. In order to complete the attachment you need to provide the following:

- A job number
- The database the application attaches to
- The name of the user logging on to the database
- The user's password

Following are the prompts to complete the attachment.

Completing the Job Number Prompt

1. At the **Job Number:** prompt type a number that does not conflict with current jobs (this should be a large number) and press Enter.

   The job number is used by Request Queue to track and update individual jobs. Because the application is being executed manually, the value of this parameter does not matter.

   Note: When specifying the Job Number, use the OFSA_REQUEST_QUEUE_SEQ to obtain the next value so that you do not overwrite any information in the Request Queue and Result Queue tables for existing jobs. To ensure that you obtain a valid Job Number that was not used by a previous job, execute the following SQL:

   select OFSA_REQUEST_QUEUE_SEQ.nextval from dual;

Completing the Database Prompt

1. Look up the database alias for the target database in the ofs.ini file. This file is located in the OFSA_INSTALL/etc directory on the server.
2. At the **Database:** prompt type the database alias and press Enter.

Completing the User and Password Prompts

1. At the **User:** prompt type in your user name for the target database and press Enter.
2. At the **Password:** prompt type in your password and press Enter. The typed characters do not appear in the window.

Setting the Application-specific Arguments

Application-specific arguments for Balance & Control, Transfer Pricing, Performance Analyzer, and Risk Manager are described as follows.

When entering names you can specify a group name in the following format [ID NAME].[FOLDER NAME]. If a group name is not used, the <ALL> group is assumed.

Balance & Control

The server-side of this application requires two additional items of information before it can be run:

- The System ID number or name of the Processing ID
- The Configuration ID or name associated with the Processing ID

Completing the Process ID Number Prompt

1. At the **`Process ID:`** prompt type in the System ID number or name for the selected Processing ID and press Enter.

Completing the Configuration ID Prompt

1. At the **`Configuration ID:`** prompt type in the System ID number or name associated with the Configuration ID and press Enter.

Performance Analyzer

The server-side of this application requires four additional items of information before it can be run.

- The System ID number or name of the selected Processing ID
- The ID type (type of process to run)
- A specified as-of-date
- A designation for the preview flag

Completing the Allocation ID Prompt

1. At the **`Allocation ID:`** prompt type in the System ID number or name for the selected Processing ID and press Enter.

Completing the ID Type Prompt

1. Type **`0`** [zero] and press Enter. No other option is available.

Completing the As-Of-Date Prompt

1. Determine the as-of-date you want to use for the Processing ID. The date format is mm/dd/yyyy.
2. At the **`As of date:`** prompt type in the desired date and press Enter.

Completing the Preview Flag Prompt

Decide whether or not you want to preview the results before actually running the process and writing data back to the table.

1. Set the preview flag to **1** to preview without changing data. Set the preview flag to **0** [zero] to change data without previewing.
2. At the **Preview flag:** prompt type the appropriate selection and press Enter.

Transfer Pricing

The server-side of this application requires the following two additional items of information before it can be run:

- The System ID number or name for the selected Processing ID
- System ID number for the associated Configuration ID

Completing the ProcSysID Prompt

1. At the **ProcSysID:** prompt type in the System ID number or name for the selected Processing ID and press Enter.

Completing the ConfigSysID Prompt

1. At the **ConfigSysID:** prompt type in the System ID number or name associated with the Configuration ID and press Enter.

Risk Manager

The server-side of this application requires the following two additional items of information before it can be run:

- The System ID number or name for the selected Processing ID
- The System ID number or name for the associated Configuration ID

Completing the ProcSysID Prompt

1. At the **ProcSysID:** prompt type in the System ID number or name for the selected Processing ID and press Enter.

Completing the ConfigSysID Prompt

1. At the **ConfigSysID:** prompt type in the System ID number or name associated with the Configuration ID and press Enter.

Troubleshooting

The server releases of the OFS applications are designed to perform batch processing. Because of this orientation, direct feedback from the server application to the client is minimal. In the case of troubleshooting error messages are written to the Request Queue log file.

To identify and correct problems within Request Queue you need to be able to interpret log file entries, recognize error categories, understand the database structure and functionality and be knowledgeable in three-tier client / server architecture.

Interpreting the Log File

Two categories of entries are written to the Request Queue log file. These are:

- Process tracking records
- Status and error messages

The following table provides server-side return code values and corresponding descriptions.

Return Code Value	Return Code Description
1000	The ofs.ini file could not be found.
1001	The application started successfully.
1002	Request Queue was unable to fork off the child application. Refer to the logfile for more information about why the fork failed.
1003	The job (application's execution) was canceled by the user.
1004	Requesting Request Queue to execute the specified application.
1005	The application completed its execution successfully.
1006	The arguments passed to the application were incorrect (bad usage).
1007	The application was unable to attach to an additional session.
1008	The application was unable to allocate additional memory.
1009	An error message was printed by the application in the log file or was output into one of the error tables (OFSA_MESSAGE_ LOG or OFSA_PROCESS_ERRORS).

Return Code Value	Return Code Description
1010	The application failed to connect to the DBMS system.
1011	The user does not have the appropriate rights to execute the specified application.

Process Tracking Records

There are two sub-categories within the process tracking records categories. These are:

- Process startup information records
- Process completion records

Process Startup Information Records

The following is an example of a process startup information record:

```
Sun Aug 13 08:50:37 1995:  Job: 278 request PING user <username>priority 1 app
                           id 0 host
Sun Aug 13 08:50:37 1995:  Responding to ping request
Sun Aug 13 08:50:43 1995:  Job: 278 request BC user BL priority 1 app id 4 host
Sun Aug 13 08:50:43 1995:  Job number 278 started with process id 2825
```

This record indicates that:

- Each transaction logged on the tracking record received a date and time stamp.
- The client application pinged Request Queue to see if it was listening.
- Request Queue responded, verifying it was functional.
- The client application issued a Balance & Control (BC) request (execute server BC).
- The Balance & Control request was initiated as job number 278 and UNIX process ID (pid) number 2825.

Process Completion Records

Process completion record is the second type of tracking record. An example of this type of record follows:

```
Tue Aug 8 20:26:39 1995:   Process id 16694 completed with return code 1005
```

This record indicates that the operating system assigned a pid of 16694 and the process completed its execution with a return code of 1005, indicating a successful completion.

Either the job number or pid is needed to verify that a task completed properly. Review the process startup record or query the TSER_REQUEST_QUEUE table to locate both the job number and the pid.

Status and Error Messages

Each spawned process writes status information and error messages to the log file, grouped in the following categories:

- Application startup status messages
- Message box status messages
- Error messages

Application Startup Status Messages

An application startup status message provides OFSA release information, the application being executed and application-specific arguments. An example of an application startup message appears:

```
Oracle Balance and Control
Version 4.0
Copyright 1995-1998 Oracle Corporation
All rights reserved Worldwide
Running Balance and Control:  Process ID: 105236 Config ID: 104900
```

This record indicates that Release 4.0 of Balance & Control is being executed and that the Balance & Control process being run is Process ID 105236 using Configuration ID 104900.

Message Box Status Messages

These messages provide specific, detailed information about the launched process, including the application that generated the message. Note, however, that neither the process job number nor pid are included in this information.

The text contained in a message box is the same whether the process is launched on the server or on a client.

An example of a message box status message appears:

```
MB(/BC): SQL Stmt: select IDENTITY_CODE, ID_NUMBER, DATA_SOURCE, GL_ACCOUNT_ID,
CREDIT_STATUS_CD, REPRICE_FREQ, REPRICE_FREQ_MULTI, INTEREST_RATE_CD, DIRECT_
IND_CD, AMRT_TYPE_CD, LOAN_TYPE, TP_COA_ID, COLLATERAL_CD, TAX_EXEMPT_PCT from
CONSUMER_LOAN where (CONSUMER_LOAN.DATA_SOURCE = '26')))
```

Error Messages

These messages include the error, in both text and code values, the application that generated the error and the job number. Job numbers are assigned sequentially, beginning with 1. The numbering scheme is reinitialized when the OFSA_REQUEST_QUEUE is flushed.

An example of an error message follows:

```
E(/TP[1])50 (203105) OFS Oracle drv_oci Error:
OCI Function:   [0] - olon(), orlon()
Oracle Error:   [1017] - ORA-01017: invalid username/password; logon denied
Driver Function:        drv_oci::Connect()
```

This error message indicates that the application generating the message is Balance & Control and the job number is 226.

Types of Errors Written to the Log File

Six categories of errors are written to the log file. These include:

- Application errors
- Asynchronous and concurrent processing errors
- Database errors
- Improper usage errors
- SQL syntax errors
- System resource errors

Each of these categories are discussed.

Application Errors

This is an application-ending error such as a core dump. If this occurs contact Oracle Support Services.

Asynchronous or Concurrent Processing Errors

Generally, these errors are collisions or deadlocks caused by row, page or table-level locks in a DBMS system. This problem should be referred to Oracle Support Services.

Database Errors

Problems with database definition, data integrity or internal errors in the system vendors or driver (OCI for example) will cause database errors. The most common are data integrity errors. The reason for this is that the OFS applications provide users with the ability to define many assumptions and classification methods. This flexibility can lead to errors such as a variable overflow when retrieving a value from the Oracle database. If the source of the database error is either a database definition or an internal error contact Oracle Support Services.

Improper Usage Errors

The OFS applications are highly configurable, providing users with the capability of customizing application assumptions and environment. The complexity of the applications support this increased flexibility but can also lead to errors of this types if the user is not sufficiently familiar with the application processes. Look for obvious errors in the Processing ID launched by the application. If none are found contact Oracle Support Services.

SQL Syntax Errors

These types of errors are caused either by an error within the application or the OFSA Processing ID is corrupted or improperly defined.

An example of an improperly defined Processing ID would be a correction Processing ID with a bad assignment rule.

If the SQL syntax error appears contact customer support.

System Resource Errors

System resource errors with respect to the OFSA server applications occur when hardware assets are overloaded or permissions and limits for the user are inadequate.

Interpreting Server Job Return Messages

Request Queue posts a return status message to the server status window for each job initiated. The messages appearing are returned to the job return status column within the server status window.

These messages follow.

Bad Usage

The sub-process failed to start because the application parameters violated the application's argument passing requirements. If this message appears contact Oracle Support Services.

Connect Failure

The server process was unable to connect to the database. Most likely the cause is an incorrect user ID or password. This can occur if the password is changed after starting up the application but before a server process is run. Check the log file for more information.

Failed on Fork

A requested application could not be initiated by Request Queue as a child process. If one of the following conditions exists a requested application cannot be executed.

- The path to the application is either incorrect or missing.
- Each application available for execution must be located within the .INI file under the [OFSRQ] label.
- The user initiating Request Queue does not have execution permission for the requested application.
- The maximum number of processes allowed for the user who initiated Request Queue has been exceeded. Correct this problem by increasing the appropriate kernel parameter for maximum processes allowed on the system.

Internal Error

An error occurred within the server application. Normally this is a database error. Check the log file or appropriate database table (OFSA_PROCESS_ERRORS or OFSA_MESSAGE_LOG) for additional information.

Job returned: <number>

The process died due to the signal number that is shown as a negative number (for example, *Job returned: -11*). Check the log file for additional information and notify Oracle Support Services.

Making Request

A client application submitted a request to the OFSA_REQUEST_ QUEUE table, but the requested action has not been initiated.

No memory

The server application was unable to allocate additional memory.

None: canceled

A process has been canceled. A process can be terminated either by a user from within the server status window (client application) or by a UNIX signal. The signals 1, 2, or 15 (SIGHUP, SIGINTR, or SIGTERM) generates this message.

None: running

The job is running on the server.

No .ini found

Two circumstances generate this message. Either the .INI file cannot be located or the user is unauthorized to read the .INI file.

If user authorization is not the issue check the location of the .INI file. This file is located in the **etc** directory of the OFSA_INSTALL directory.

Normal

The job successfully completed.

Rights Violation

The user does not have the proper rights to run the program. This can occur if an administrator changes an individual's rights after a client application has been started but before the server application is launched.

Session Failure

The application was unable to obtain a database session. Check the server log file for additional information regarding the source of the problem.

Multi-Host Request Queue

This iteration of Request Queue is referred to as Dynamic Multi-Host Request Queue (MRQ). MRQ enables client jobs to be assigned to multiple servers by automatically distributing client jobs among the servers that are available.

Note: You cannot run MRQ and Request Queue at the same time.

MRQ operates in a master-slave mode. The master Request Queue assigns jobs to the available hosts, while the slave (or normal) Request Queues execute the jobs assigned to them. No user action is needed to create a master Request Queue and if the master Request Queue is shutdown or not responding, one of the normal Request Queues automatically promotes itself to master.

The first MRQ to be started on a database promotes itself to master. Each Request Queue will regularly update a row in the new database table RQ_STATUS with a timestamp to show that it is up and running. The master Request Queue uses this table to determine which hosts are up and should be assigned jobs.

If a Request Queue is shut down gracefully it will remove its row from this table. If a Request Queue or host crashes, its timestamp will not be updated and it will be removed after a specified amount of time (default 60 seconds). If the master Request Queue is shut down, the first Request Queue to notice that the master is missing from the RQ_STATUS table promotes itself to master. If the master Request Queue hangs or crashes, its timestamp in the table becomes old and after a specified amount of time (HostTimeOut, default 60 seconds) another Request Queue will notice this and become master. Table locking is used to ensure that two Request Queues do not become master at the same time.

Client jobs are distributed in round-robin fashion among all available hosts. If there are four new jobs and three hosts, the jobs are assigned to host1, host2, host3, and host1, respectively. Client PCs are not assigned to a particular host. The master Request Queue assigns jobs to a host by setting the HOST_NAME column of the job's entry in the OFSA_REQUEST_QUEUE table.

Note: Refer to the section entitled "Single-Host Request Queue" for detailed information on the overall Request Queue process.

Installation and Configuration

This section discusses the installation and configuration of MRQ.

Launching Dynamic Multi-Host Request Queue

DMRQ is not automatically activated when the Oracle Financial Services (OFS) applications are installed on the server and client. This is a manual process, performed by the System Administrator or DBA, using either the **mrq** command, which is a UNIX shell script designed to set the required environment variables automatically, or by launching DMRQ directly.

> **Note:** Oracle recommends that the person responsible for launching Request Queue is also responsible for maintaining the applications.

If you decide to launch DMRQ directly you need to set up the environment variables manually, according to the requirements of the server vendor. You can use the mrq script as a reference for the variables you need to address. See Chapter 6, "UNIX Server Installation and Configuration" for detailed information on setting up the server environment variables specific to the server on which you are installing the OFSA server-centric software.

The database upgrade process adds the new table, RQ_STATUS, to the database. See the section entitled "Database Changes - RQ_STATUS Table" for information on this table.

Using the mrq Script to Set Up Dynamic Multi-Host Request Queue

Refer to the section entitled "Using the rq Script to Set Up Request Queue" for detailed information on setting up the operational parameters of DMRQ. The operational parameters for DMRQ are the same as for Request Queue.

Configuring Dynamic Multi-Host Request Queue

No configuration changes are necessary to make DMRQ work. However the following parameter can be adjusted in the OFS.INI file, in the [ofsrq] section:

HostTimeOut	Time (seconds) before a host is declared down. Default setting is 60 seconds.

The MappedRQHost and DistMode parameters are no longer used and may be deleted.

The time between updates of the timestamp, and between those times, when the master assigning jobs, is controlled by the **-i** option of Request Queue. This value, the Poll Interval, has a default of 3 seconds.

The Request Queues get the name of their host from the Nodename returned by uname **-n**, the name by which the system is known on a communications network. These names should be unique.

Client Software Changes - Server Status Window

In the Server Status window, the Status column can show a new state, *Host Assigned*. This means the job has been assigned to a host but has not started executing yet. The following is a sequence of states that a job goes through at startup: Request Execution, Host Assigned, Executing. If the master Request Queue is functioning properly, a job should go from Request Execution to Host Assigned within a few seconds (controlled by the Poll Interval).

Database Changes - RQ_STATUS Table

The common communication point for the Request Queues is a table in the database called the RQ_STATUS table. There is a single row in this table for each Request Queue that is currently running. This table has the following columns:

Column Name	Description
HOST_NAME	This column stores the name of the host of each of the Request Queues. It is the Primary Index of the table and its values are unique.
RQ_STATUS	This column has the codes for Normal (0) or Master (1).
STATUS_DATE	The date column is updated on a regular basis by each of the Request Queues with a current date and time, as retrieved from the database. This timestamp represents the last time the Request Queue checked the table. The status date is in Oracle date format, which embeds both date and time. The date and time are used to determine whether a Request Queue is currently active.

One of the rows in this table is designated the master Request Queue and contains the code for MASTER in the RQ_STATUS column.

The following table shows the master and normal Request Queues in a typical RQ_STATUS table.

HOST_NAME	STATUS	STATUS_DATE
host1	Normal (0)	Mar 4 1997
host2	Master (1)	Mar 4 1997
host3	Normal (0)	Mar 4 1997
host4	Normal (0)	Mar 4 1997

Additional Multi-Host OFS.INI Parameters

The following table lists additional ofs.ini parameters for DMRQ.

Parameter	Description
ServerTimeOut	Time client waits (in seconds) for server to respond to pings. The default setting is 30 seconds.
ClientPollInterval	Time interval (in seconds) between client polls of Request Queue for the Server Status window. The default setting is 1.65 seconds.

Troubleshooting

This section provides information on troubleshooting problems in multi-host Request Queue.

Host Crashes

If a Request Queue or its host hangs or crashes, there is a short period of time (HostTimeOut) before this is recognized by the master Request Queue. During this time jobs are still assigned to that host by the master. Those jobs will run when the host and Request Queue are restarted. Alternatively, you can cancel the jobs and rerun them on other hosts.

Master Request Queue Hangs

DMRQ is designed to recover from most combinations of hosts crashing or hanging. There is one scenario, however, which cannot be handled by DMRQ. In this scenario the master hangs (but does not crash) while locking the RQ_STATUS table. This situation would give the following symptoms:

- New jobs are not being assigned to hosts. In the client's Server Status window the jobs stay in the Request Execution state and the Host column is blank.
- Some of the other Request Queues have tried to become master and then hung. Check for the message *Master XXX is old, trying to become new master* at the end of the logs.
- Master Request Queue is still running.
- Master Request Queue's timestamp in RQ_STATUS is, most-likely, not being updated.

In this circumstance you need to kill the master Request Queue manually. First try killing it by running **rq -K** on the same machine. If the master Request Queue does not acknowledge this command with *RQ Termination requested*, then kill it with the **kill** command. If that command is unsuccessful use the **kill -9** command. Use this last command as a last resort.

Debug Option

There is a debug option for Request Queue (**-d**) that outputs verbose information that will be useful to Oracle Support Services in diagnosing problems.

21

FDM Utilities

Oracle Financial Data Manager (FDM) Utilities include scripts and procedures that assist you in managing the FDM database environment. These utility scripts are installed with the FDM database package on the server. This chapter describes the purpose for these scripts and how to use them.

The following FDM Utilities are described in this chapter:

- Add Leaf
- Currency Mapping
- Changing Functional Currency
- Instrument Templates
- Ledger Stat Load
- Modify Balance Column Size
- Recompiling Packages, Procedures, and Java Classes
- Recompiling Views and Triggers
- Instrument Synchronization
- Codes Synchronization
- Reporting Utilities

The default location for the FDM utility scripts is the **OFSA_INSTALL/dbs/<OFSA release>/utilities** subdirectory of your OFSA installation directory.

> **Note:** In general, set the SQL*Plus setting SERVEROUTPUT = ON when running any FDM utilities that are invoked by executing a stored procedure. This enables the display of all status and warning messages.

Add Leaf

The Add Leaf procedure performs the following:

1. Adds the new Leaf Column name to all of the appropriate tables based upon Table Classification assignments as NUMBER(14) NOT NULL
2. Updates the new Leaf Column with a default value of 0 Registers the new Leaf Column on all affected tables as FDM Data Type = LEAF.

The ADD_LEAF procedure adds the specified column_name as NUMBER(14). It then registers the column name on the affected table as FDM Data Type LEAF.

> **Note:** The ADD_LEAF procedure only completes the first part of registering a new Leaf Column for the FDM database. There are additional steps required to complete the Leaf Registration. Refer to Chapter 17, "FDM Leaf Management"for complete instructions on registering a new Leaf Column.

Use the ADD_LEAF procedure to add the new Leaf Column to all of the required tables. However, this procedure does not affect views. You must manually drop and re-create any views of the designated Table Classifications so that they include the new Leaf Column.

Tables Affected by ADD_LEAF

The list of objects altered by the ADD_LEAF procedure is based upon the specified Leaf Type and existing Table Classification assignments. Tables with the following Table Classification assignments, where the Leaf Column does not already exist on the object, are altered by the ADD_LEAF procedure to include for new All (Both Instrument and Ledger) Leaf Columns:

Tables affected for Leaf Type = Both Instrument and Ledger

Table Classification CD	Description
50	Ledger Stat
200	TP Cash Flow
210	TP Non-Cash Flow
300	Transaction Profitability
310	Instrument Profitability
351	Requires all B Leaves
360	RM Standard
370	TP Option Costing

For Leaf Columns of Registration Type Ledger Only, the Leaf Column must exist on each object registered for the following Table Classifications:

Tables affected for Leaf Type = Ledger Only:

Table Classification CD	Description
50	Ledger Stat
352	Requires all L Leaves

If the Leaf Column already exists on a table, the ADD_LEAF procedure ignores that table and does not attempt to add (or register) the new Leaf Column to it. Only objects in the Table Classification assignments that are of OBJECT_TYPE = TABLE are altered by the procedure.

> **Note:** While Views also require the new Leaf Column, the ADD_LEAF procedure does not affect them. Drop and recreate any required views manually to add the new Leaf Column to them.

Running the Procedure with the RUN_ADD_LEAF script

To run the ADD_LEAF procedure, go to the **OFSA_INSTALL/dbs/<OFSA release>/utilities/add_leaf** directory. Then, login to SQL*Plus as the FDM Schema Owner and type the following command:

```
SQL> @run_add_leaf
```

This script prompts you for the following parameters:

```
Leaf Column Name
Display Name
Leaf Type
DBF Name
```

These parameters are described as follows:

Leaf Column Name This is the column name of the new Leaf Column. The name is restricted to <=30 characters.

Display Name This is the name for the column as it appears in the OFS applications. The Display Name is restricted to <=40 characters.

Leaf Type The Leaf Column must be identified as *L* - Ledger Only or *B* - Both Instrument and Ledger. The Leaf Type determines the object on which the new column is added.

DBF Name This is the name of the new column for database file exports. It is restricted to <=10 characters.

Calling the Procedure Directly

If you want to call the procedure directly, instead of using the run_add_leaf script to prompt for the required parameters, execute the following statement as the FDM Schema Owner:

```
SQL> EXECUTE ofsa_util.add_leaf('&db_owner', '&leaf_column', '&display_
name', '&leaf_type', '&dbfName');
```

Provide the parameters.

Reviewing ADD_LEAF DDL and DML statements

The ADD_LEAF procedure outputs all DDL statements to the OFSA_STP table for TASKNAME = ADD_LEAF. It also outputs the DML statements for setting the default value of the new Leaf Column = 0 with TASKNAME = UPDATE_LEAF. Review the statements in this table for these TASKNAME values to verify that all statements were processed successfully. For DDL statements (ADD_LEAF), a STATUS = 0 indicates success, while any other number designates failure with the

STATUS value being the Oracle error. For DML statements (UPDATE_LEAF), the STATUS indicates the number of rows updated by the procedure.

Currency Mapping

The Currency Mapping utility enables you to map values from the old CURRENCY_CD column to the new ISO_CURRENCY_CD standard values in your Instrument tables. Previously, the CURRENCY_CD column enabled users to report instrument data in different currencies. However, the code values for this column were not standardized. FDM 4.5 replaces CURRENCY_CD with ISO_CURRENCY_CD column for multi-currency functionality. The code values for this new column are based upon ISO standards and are seeded by the FDM database creation and database upgrade processes.

FDM defaults the ISO_CURRENCY_CD value for all Instrument table records to the Functional Currency specified during the database upgrade process. FDM preserves the old CURRENCY_CD column so that you can use the Currency Mapping utility to map instrument records to the new values. Run this procedure after the FDM 4.5 database upgrade process is complete.

Complete the following steps to run this procedure:

- Creating the Mapping Table
- Defining Mapping Assignments
- Executing the Procedure
- Review Log Information

Creating the Mapping Table

The Mapping Table defines how the old CURRENCY_CD values are mapped to the new ISO_CURRENCY_CD values. The Currency Mapping procedure reads this table to perform the appropriate update statements on the instrument tables.

From the **OFSA_INSTALL/dbs/<OFSA release>** directory login to SQL*Plus as the FDM Schema Owner and run **currencies_mapping_setup.sql** to create the Mapping table. This script takes the following input parameters:

<Password> This is the password for the database owner, to invoke SQL*Loader.

<Upgrade home dir> This is the full path to the database upgrade home directory as previously described. Do not enter a trailing / character.

An example of a full path follows:

```
/app/ofsa/ofsa4.0-092/dbs/400001010
```

<Sql*Loader executable> This is the command used to execute SQL*Loader. For most Oracle installations, this command is sqlldr.

If you run **currencies_mapping_setup.sql** with no command line parameters, it prompts you for them. It then prompts you to confirm that the parameters you entered are correct.

```
SQL> @utilities/convert_currencies/currencies_mapping_setup
```

Or:

```
SQL> @utilities/convert_currencies/currencies_mapping_setup <Password>
<Upgrade home dir> <Sql*Loader executable>
```

The **currencies_mapping_setup.sql** exits SQL*Plus with an error message under the following conditions:

- You type N at the confirmation prompt.
- Any of the input parameters you entered are invalid.

After running the script, refer to the currencies_mapping_setup.log for error messages. This log file is in the **OFSA_INSTALL/dbs/<OFSA release>/log** directory. Ignore any *Table or View does not exist* errors occurring during Drop Table statements.

Defining Mapping Assignments

The mappings of old CURRENCY_CD values to new ISO_CURRENCY_CD values are defined in the OFSA_TEMP_CURRENCIES_MAPPING table. This table is created by the currencies_mapping_setup script.

FDM provides a set of default mappings for this table. However, because previous versions of OFSA did not enforce any standards for CURRENCY_CD values, it is possible that the default mappings do not reflect your organization's implementation. Review the mappings in this table to verify that they are correct for your implementation.

The structure of this table is as follows:

Column Name	Description
CURRENCY_CD	Identifies the old CURRENCY_CD values for OFSA 3.5/4.0.

Column Name	Description
ISO_CURRENCY_CD	Designates the new ISO_CURRENCY_CD value for FDM 4.5.

FDM provides the following default mappings. Modify these mappings as appropriate for your implementation:

CURRENCY_CD	ISO_CURRENCY_CD
110	CAD
120	GBP
130	DEM
140	CHF
150	FRF
160	ITL
170	JPY
180	MXP
200	AUD
210	ARA
220	BHD
230	BEF
240	BRC
250	CSK
260	CLP
270	CNY
280	COP
290	DKK
300	ECS
310	FIM
320	GRD
330	HUF

CURRENCY_CD	ISO_CURRENCY_CD
340	HKD
350	INR
360	IDR
370	IEP
380	ILS
390	JOD
400	KWD
410	LBP
420	MYR
430	MTL
440	ANG
450	NZD
460	NOK
470	PKR
480	PEI
490	PHP
500	PLZ
510	PTE
520	SAR
530	SGD
550	ZAR
560	KRW
570	ESP
580	SEK
590	TWD
600	THB
610	TRL
620	AED

CURRENCY_CD	ISO_CURRENCY_CD
630	UYP
640	VEB

Executing the Procedure

After editing the Currency Mapping table, run the procedure to update designated Instrument tables with the new ISO_CURRENCY_CD values. To do this, login to SQL*Plus as the FDM Schema Owner and type the following:

```
<SQL> execute ofsa_convert_currencies(p_table_name);
```

where p_table_name is the Instrument table name that is updated by the procedure. For example:

```
<SQL> execute ofsa_convert_currencies('MORTGAGES');
```

In this example, the procedure updates the MORTGAGES table, setting the ISO_CURRENCY_CD value equal to the specified value from the Currency Mapping table based upon the *old* value in CURRENCY_CD.

To update all Instrument tables, type the following:

```
<SQL> execute ofsa_convert_currencies('ALL');
```

This statement updates all tables with the *Instrument* Table Classification (table_classification_cd=20) on which both the CURRENCY_CD and ISO_CURRENCY_CD columns exist.

Review Log Information

Review the OFSA_STP table for information regarding update failures. This utility outputs messages to the OFSA_STP table where the TASKNAME column = convert_currencies.

Changing Functional Currency

Functional Currency is defined as the currency of the primary economic environment in which an entity conducts its business. Both the FDM database upgrade and database creation processes prompt for a Functional Currency. FDM provides a procedure named **set_default_currency** for changing the Functional

Currency after it has already been set by the FDM database upgrade or database creation processes.

When you specify the Functional Currency during the database upgrade or creation processes, the FUNCTIONAL_CURRENCY_CD column in OFSA_DB_INFO is updated with the designated value. The ISO_CURRENCY_CD column in the Instrument and LEDGER_STAT tables is also updated with the specified Functional Currency. In addition, all *Instrument* tables on which the ISO_CURRENCY_CD column exists are altered to default to the Functional Currency for any inserts in which an ISO_CURRENCY_CD value is not specified.

Changing the Functional Currency for an FDM database therefore involves the following:

- Updating OFSA_DB_INFO
- Updating Instrument and LEDGER_STAT tables
- Running the SET_DEFAULT_CURRENCY procedure

Each of these steps is described as follows:

Updating OFSA_DB_INFO

The OFS applications access the OFSA_DB_INFO table to determine the Functional Currency for a database. To change the Functional Currency, execute the following SQL statement as the FDM Schema Owner:

```
update ofsa_db_info
set functional_currency_cd = :new_currency;
```

Updating Instrument and LEDGER_STAT Tables

If you are changing the Functional Currency for your database, you may need to update data in your Instrument and LEDGER_STAT tables. Review all Client Data Objects with the ISO_CURRENCY_CD column and update as appropriate.

Running SET_DEFAULT_CURRENCY

The SET_DEFAULT_CURRENCY procedure alters all Instrument tables and LEDGER_STAT so that they default to the new Functional Currency for any insert statements where ISO_CURRENCY_CD is not explicitly specified. This involves creating a default clause on these tables. To change the Functional Currency, execute the following SQL statement as the FDM Schema Owner:

```
execute ofsa_dba.set_default_currency(USER_NAME);
```

For example:

```
execute ofsa_dba.set_default_currency('SCHEMA_OWNER');
```

This procedure sets the default for ISO_CURRENCY_CD for all Instrument and LEDGER_STAT tables with the value specified in the FUNCTIONAL_CURRENCY_CD column in OFSA_DB_INFO.

Instrument Templates

FDM provides template scripts to facilitate creating and altering Instrument tables. These templates incorporate standard definitions for FDM Reserved columns to make it easier for you to create tables with all of the columns required by FDM Table Classifications.

The Template scripts are commented with instructions on how to remove FDM Reserved columns and where to add user-defined columns to the script. Because all Client Data tables are customizable, the templates are setup to make it easy for you to add and remove columns as desired.

The Instrument Template scripts are located in the **OFSA_INSTALL/dbs/<OFSA release>/utilities/instrument_templates** directory. Refer to the comments included in each template script for information on modifying these scripts for your use.

FDM provides the following template scripts:

Script Name	Description
add_inst.sql	Template for adding FDM Reserved columns required by Table Classifications to a table.
add_optional_profit.sql	Template for adding optional profitability columns to a table. These columns are not required by any Table Classifications, but are optional fields for profitability processing.
add_portfolio.sql	Template for adding fields for the standard Portfolio Table Classification to a table. Because the Portfolio classification is customizable, you need to add/remove columns from this template as appropriate.

Script Name	Description
cr_foward_contracts	Template for creating the new FORWARD_ CONTRACTS table. This table is included as part of the FDM data model for a new installation (database creation process) but is not automatically added to a database upgraded from OFSA 3.5/4.0.
cr_inst.sql	Template for creating a generic Instrument table for OFS processing operations.
cr_interest_rate_options	Template for creating the new INTEREST_ RATE_OPTIONS table. This table is included as part of the FDM data model for a new installation (database creation process) but is not automatically added to a database upgraded from OFSA 3.5/4.0.
cr_interest_rate_swaps	Template for creating the new INTEREST_ RATE_SWAPS table. This table is included as part of the FDM data model for a new installation (database creation process) but is not automatically added to a database upgraded from OFSA 3.5/4.0.
cr_term_deposits	Template for creating the new TERM_ DEPOSITS table. This table is included as part of the FDM data model for a new installation (database creation process) but is not automatically added to a database upgraded from OFSA 3.5/4.0.
cr_trans.sql	Template for creating a generic Transaction Profitability table.

Ledger Stat Load

The Ledger_Stat load utility is an Oracle stored procedure used to load your ledger data into the Oracle Financial Services Applications (OFSA) LEDGER_STAT table.

The following topics are included in this section:

- Features of the load procedure
- Overview of the load procedure
- Setup for the load utility
- Running the procedure

The Ledger_Stat load procedure can be invoked from either the OFSA software or directly from the server. The process itself runs on the server.

Note: The client-side Ledger Load is not longer supported for FDM version 4.5.

Features

The Ledger_Stat load utility is the only supported method for loading your ledger data into the LEDGER_STAT table. It is compatible with the Import Ledger feature previously available in Performance Analyzer, with the following differences:

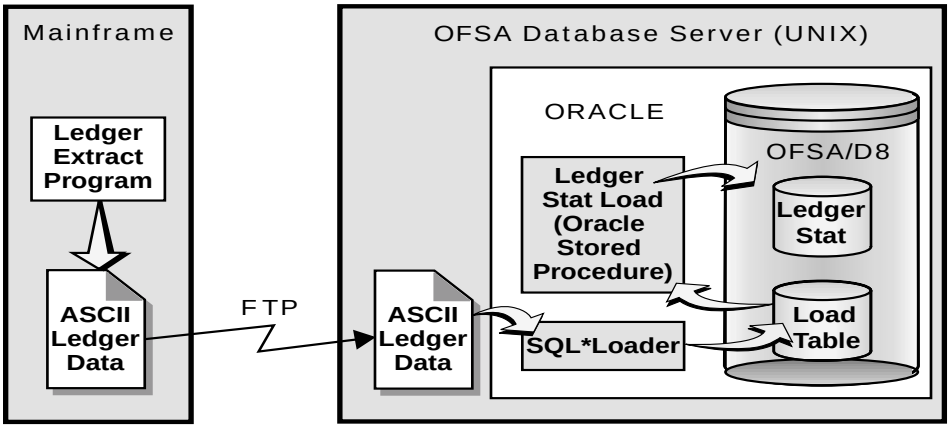
- The Ledger_Stat load utility loads data from an Oracle table, rather than from a .dbf file. You use SQL*Loader to load your ASCII ledger data into one or more load tables.
- The Ledger_Stat load utility runs on the server rather than the client, and uses a bulk SQL approach rather than a row-by-row approach. It is therefore significantly faster than the Import Ledger feature of Performance Analyzer, and is recommended for large Ledger_Stat loads.
- New leaf values are not added to the Leaf Setup tables or the Tree Rollups by the Ledger Stat load utility. This is done separately by the Synchronize Instrument Utility. See Executing the SYNCHRONIZE_INSTRUMENT Procedure for more details on this topic.

The Ledger_Stat load utility offers the following features:

- In addition to loading one month or all months, you can specify a range of month columns to be loaded.
- A month can be undone individually, using the Undo feature in Performance Analyzer. You can do this even though the month to be undone was included in a multiple-month load.
- You can update columns in existing Ledger_Stat rows using either the additive or replacement functionality.
- You can bypass the upsert logic and insert all the rows from the load table using the INSERT_ONLY mode. This functionality can be used either for first-time loads or to reload for all months with each load.
- You can run the load procedure as a batch load of multiple load tables, either sequentially or in multiple, concurrent processes.

Overview of the Load Process

The following diagram illustrates the process flow for the Ledger_ Stat load procedure.



In the load procedure ASCII ledger data is loaded into an Oracle table (resembling the ASCII file) using SQL*Loader. Script templates are provided to create the load table(s).

Runtime parameters, such as the name of the load table, which columns to load, ADD or REPLACE update functionality, and whether or not to create offset records can be entered into the Ledger Stat Load Batch table using a Data Verification ID. One row per table to be loaded is entered in this table, as a way to batch a multiple-table load in one run of the procedure.

The procedure is implemented as an Oracle PL/SQL stored procedure so it can be invoked from a SQL ID or from SQL*Plus. Input parameters are read from the batch/parameter table and validated for correctness, completeness and consistency before the load begins. Parameter errors are written to a Message column in the batch/parameter table. Runtime statistics are written to the batch/parameter record following completion of the load for that record.

Limitations

The following limitations should be kept in mind.

Load Table Rows Must Be Unique

A restriction imposed by the use of bulk SQL (as opposed to row-by-row) processing is that all the rows in the load table(s) must be unique. This means that there is one row in the load table for one row in Ledger_Stat. A unique index is created on each load table to enforce this uniqueness and provide acceptable performance.

Defining Financial Elements in Leaf Setup

Occasionally, your load table may contain leaf values for one or more leaf columns that are not defined in Leaf Setup. The Ledger_Stat load procedure loads these rows anyway, except for the rows containing undefined or incompletely defined FINANCIAL_ELEM_ID values.

Any new values for FINANCIAL_ELEM_ID must first be defined in Leaf Setup before running the load. Specifically, the load procedure needs the AGGREGATE_METHOD value for each FINANCIAL_ELEM_ID value so that the YTD columns in Ledger_Stat can be computed using the appropriate method.

Setup for the Ledger_Stat Load Utility

The following table describes the installation components of the Ledger_Stat load procedure.

File	Description
lsview.sql	Creates a view on Ledger_Stat. This view is required by the load procedure. Run this script once.
lsldtbl.sql	Creates a load table, as well as an index and two views on that table. Run once for each load table you want to create.
lsload.ctl	A sample SQL*Loader control file for loading the ASCII ledger data into the load table.

These files are located in the OFSA_INSTALL/dbs/<OFSA release>/utilities/ls_load directory. In this chapter, OFSA_INSTALL is the convention used to indicate where the OFSA software is installed in your directory structure. Copy the these files to a work area and edit the copies.

The following installation steps are addressed in this section:

- Customizing **lsview.sql** to add any user-defined leaf columns that have been created in Ledger_Stat.
- Customizing **lsldtbl.sql** and **lsload.ctl** to add any user-defined leaf columns, and to add the m1-m12 monthly amount columns to the definition of the load table if you plan to load more than one month at a time.
- Running **lsview.sql** from SQL*Plus to create a view on Ledger_Stat.
- Running **lsldtbl.sql** in SQL*Plus once for each load table that you want to create.

Each of the steps involved in setting up and running the Ledger_Stat load procedure are discussed in detail in the following subsections. The setup activities need to be done only once.

In the examples that follow, Ledger_Stat has a user-defined, primary key leaf called **tp_coa_id** and a non-key leaf called **branch_org_id**. Also, 12 monthly amount columns are used.

Customizing lsview.sql

This script creates a view on the LEDGER_STAT table called LSL. The purpose of this view is to provide shorter column names for the load procedure. The LSL view must contain the same columns as Ledger_Stat.

For any user-defined leaves in your Ledger_Stat you must complete the following steps.

1. In FDM Administration, look up the DBF_NAME for that leaf column in the Object Columns tab.
2. In the lsview.sql file, Go to the comment line "-- Other leaf columns:" (near the end of the CREATE or REPLACE VIEW statement) and add a line for each user-defined leaf in your LEDGER_STAT table, using the DBF_NAME for that leaf.

   Be sure that the commas are accurate. Every line requires a comma except the line immediately before the FROM clause.

An example of a customized lsview.sql file follows:

```
            ytd_10                  AS y10,
            ytd_11                  AS y11,
            ytd_12                  AS y12,

-   Other leaf columns:
-      [Real name in L/S]           AS [DBF_NAME from OFSA_TAB_COLUMNS]

            tp_coa_id
            branch_org_id           AS branch_id

FROM ledger_stat;
```

Customizing lsldtbl.sql

This script, when run, prompts you for the name of a load table to be created. It then creates an Oracle table by the name you specified, as well as a unique index on the table and two views, which are used by the load procedure. Complete the following steps to perform this procedure.

1. Go to the CREATE TABLE statement. This statement creates a load table similar in structure to the .DBF load table used with the System Administration Import Ledger feature.

   The m1-m12 columns are optional. If you plan to use the load procedure to load more than one column at a time, then uncomment these lines (space over the "--"). The **one_month_amt** column corresponds to the **cur_book_b** column in the .DBF load table.

2. Go to the comment line"-- Other leaf columns:." After this line, add a line for each user-defined leaf in your LEDGER_STAT table.

3. Edit the INITIAL and NEXT storage parameters if you expect to load smaller (under 2 MB) or larger (over 50 MB) amounts of data.

   The current settings of 2 MB INITIAL and 2 MB NEXT are adequate (but not optimal) for loading nearly 1 gigabyte (GB) of data. Consult with your DBA regarding the appropriate storage parameter settings.

An example of a customized CREATE TABLE statement follows.

Note that **tp_coa_id** is NOT NULL because it is part of the primary key, as are all key leaves.

```
. . .

    m10NUMBER(15,4),
    m11NUMBER(15,4),
```

```
    m12NUMBER(15,4),
--
    one_month_amtNUMBER(15,4),
--
-- ------------------------------------------------------------------------
-- Other leaf columns (DBF_NAMEs from OFSA_TAB_COLUMNS for LEDGER_STAT):
-- ------------------------------------------------------------------------
    tp_coa_idNUMBER(14) NOT NULL,
    branch_idNUMBER(14)
--
)
-- Fill in your own INITIAL and NEXT parameters with appropriate values.
-- Keep PCTFREE as 0 for best performance.
--
STORAGE ( INITIAL 100M NEXT 10M MINEXTENTS 1 MAXEXTENTS 505 PCTINCREASE 0 )
PCTFREE 0  TABLESPACE DATA_TS;
```

4. Go to the CREATE UNIQUE INDEX statement in lsldtbl.sql. This statement creates the unique index that is required for the load table. The unique key of the load table must be identical to the unique key of Ledger_Stat, with the exception that instead of IDENTITY_CODE, which is in Ledger_Stat, the load table has a column called DS (Data Source).

   a. Query the Oracle RDBMS ALL_INDEXES and ALL_IND_COLUMNS catalogs to determine the columns that compose the unique index for the LEDGER_STAT table.

   b. Determine the DBF_NAME for each column in the unique index. You can determine this by query the FDM Objects - Object Columns tab in FDM Administration. Alternatively, query OFSA_TAB_COLUMNS_V for table_name=LEDGER_STAT.

   c. Enter the DBF_NAMEs for these columns between the parentheses in the CREATE UNIQUE INDEX statement. You need to add the primary-key, user-defined leaf columns to the columns already in the list.

5. Edit the INITIAL and NEXT storage parameters if you expect to load smaller (under 2 MB) or larger (over 50 MB) amounts of data.

   An example of a customized CREATE UNIQUE INDEX statement follows. BRANCH_ID is not part of the index because it is a non-key leaf.

```
CREATE UNIQUE INDEX &ltname
    ON &ltname ( ds,
```

```
        year_s,
        accum_type,
        consolidat,
        isocrncycd,
        financ_id,
        org_id,
        gl_acct_id,
        cmn_coa_id,
    -- ------------------------------------------------
    -- Include all additional LEDGER_STAT primary key
    -- leaf columns here (use DBF_NAME from OFSA_TAB_COLUMNS_V):
    -- ------------------------------------------------
        tp_coa_id
    --
        )
-- ---------------------------------------------------------------------
-- Fill in your own INITIAL and NEXT parameters with appropriate values.
-- Keep PCTFREE as 0 for best performance.
-- ---------------------------------------------------------------------
STORAGE ( INITIAL 20M NEXT 5M MINEXTENTS 1 MAXEXTENTS 505 PCTINCREASE 0 )
PCTFREE 0  TABLESPACE INDEX_TS;
```

6. Go to the first CREATE or REPLACE VIEW statement in lsldtbl.sql. This statement creates a view on the newly-created load table. The purpose of this statement is to translate null amounts to 0, and to ignore rows where all amount columns are null or 0.

If you plan to load more than one month at a time, then uncomment (space over the "--") all the rows that reference columns m1-m12.

7. Go to the "-- Other leaf columns (DBF_NAME from OFSA_TAB_COLUMNS_V):" comment line and add a line for each user-defined leaf in your Ledger_Stat. Remember to use the DBF_NAME for each leaf.

An example of a customized CREATE or REPLACE VIEW statement follows.

```
. . .
    NVL(m10,0) AS m10,
    NVL(m11,0) AS m11,
    NVL(m12,0) AS m12,
--
    NVL(one_month_amt,0) AS one,
--
-- ------------------------------------------------------------
-- Other leaf columns (DBF_NAME from OFSA_TAB_COLUMNS_V for LEDGER_STAT):
-- ------------------------------------------------------------
```

```
    tp_coa_id,
    branch_id
--
FROM &ltname
WHERE NVL(one_month_amt,0) <> 0;
--
--OR NVL(m1,0) <> 0
--OR NVL(m2,0) <> 0
--OR NVL(m3,0) <> 0
. . .
    ;
```

8. Go to the second CREATE or REPLACE VIEW statement in lsldtbl.sql. This statement creates a second view on the newly-created load table to be used in creating offset records. This statement joins to OFSA_DETAIL_LEAVES to get the Offset Common COA and the Offset Org associated with the Common COA value in each row. This statement then returns these values in place of the Common and Org values from the input row from the load table. It ignores rows for which no offset is defined in OFSA_DETAIL_LEAVES.

   For each row returned by the view, it multiplies all the amount columns by -1.

   If you plan to load more than one month at a time, uncomment all of the lines that reference m1- m12.

9. Go to the line "-- Other leaf columns (DBF_NAME from OFSA_TAB_ COLUMNS_V):" and add a line for each user-defined leaf. Remember to use the DBF_NAME for each leaf.

An example of this second CREATE or REPLACE VIEW statement follows.

Because branch id is a non-key leaf, it is not included in the GROUP BY. This requires a group function, such as MIN or MAX, to be used on that column in the SELECT list.

```
    SUM(m10)*-1 AS m10,
    SUM(m11)*-1 AS m11,
    SUM(m12)*-1 AS m12,
--
    SUM(one)*-1 AS one,
--
-- ------------------------------------------------------------
-- Other leaf columns (DBF_NAME from OFSA_TAB_COLUMNS_V):
-- (Use the MIN() function on non-key leaves,
--  to avoid including them in the GROUP BY)
-- ------------------------------------------------------------
    tp_coa_id,
```

```
    MIN(branch_id)
--
FROM &ltname._v lt, detail_leaves dl
WHERE dl.leaf_node = lt.cmn_coa_id
AND dl.o_coa_id NOT IN (-99100,0)
GROUP BY ds,
        year_s,
        accum_type,
        consolidat,
        isocrncycd,
        financ_id,
        dl.o_org_id,
        gl_acct_id,
        dl.o_coa_id,
        -- ------------------------------------------------------
        -- Include all Primary Key leaf columns
        -- (using DBF_NAME from OFSA_TAB_COLUMNS_V) in the GROUP BY clause
        -- ------------------------------------------------------
        tp_coa_id
        ;
```

Customizing lsload.ctl

Make a copy of this file for each load table that you want to load, then follow these steps to customize the file.

1. Enter the full path name and file name of the ASCII ledger data file from which you want to load.
2. Enter the name of the load table to which you want to load.
3. Update the column position definitions to match the column positions in your ASCII load file.
4. Add a line for each user-defined leaf column. Remember to use the DBF_ NAME, as in the load table created by **lsldtbl.sql**.
5. Add lines for columns m1-m12 if you plan to load more than one column at a time.

> **Note:** The m1-m12 columns in the load table correspond directly to the MONTH_01 - MONTH_12 columns in Ledger_Stat. For example, if MONTH_01 in your fiscal period corresponds to March, then the m1 column in the load table must contain data from March because it will be loaded directly into MONTH_01 in Ledger_Stat.

Be sure to retain the “truncate into” clause. This automatically deletes all rows that populated the load table from the previous load.

Running lsview.sql

To create the lsview on Ledger_Stat you need to run the lsview.sql script. Run the script by following these steps:

1. On the server, log into SQL*Plus using the database owner user id. You need the encrypted password.
2. At the SQL> prompt type the following:

   ```
   SQL> @lsview.sql
   ```

   You only need to run this script once. However, if you need to change the script you can run it again.

Running lsldtbl.sql For Each Load Table

Run **lsldtbl.sql** for each of the load tables you plan to use in loading Ledger_Stat. As the script runs, it prompts you for a table name, which you need to enter. The script then creates the table, a unique index and two views.

To run the **lsldtbl.sql** script follow these steps:

1. On the server, log into SQL*Plus as the FDM Schema Owner.
2. At the SQL> prompt enter the following statement:

   ```
   @lsldtbl.sql
   ```

3. When prompted, enter the table name and press Enter.

An example of the **lsldtbl.sql** script run for **my_load_table** follows. Run this script for each of the load tables you plan to use.

```
SQL> @lsldtbl.sql
Load Table Name (20 chars or less): my_load_table
Table created.
Index created.
View created.
View created.
SQL>
```

If you need to change the script and run it again, using the same load table name, the script automatically drops the table created in the previous run then recreates it, and the views as well.

Running the Ledger_Stat Load Procedure

The following topics are addressed in this section.

- Steps associated with the monthly Ledger_Stat load procedure
- Running concurrent loads with multiple load tables
- Undoing Ledger_Stat load updates
- Using the Update Mode, Insert Only and Create Offsets parameters
- Troubleshooting the load procedure

The Monthly Ledger_Stat Load Process

The following topics are addressed in this section.

- Loading the ASCII ledger data into Oracle load table(s) using SQL*Loader
- Editing the Ledger Stat Load Batch parameter table using the Data Verification ID
- Invoking the load procedure
- Running the Synchronize Instrument utility if your load table includes new values in any of the leaf columns.

Loading the ASCII Data

Load each ASCII ledger data file into an Oracle load table (which you created with the lsldtbl.sql script) using SQL*Loader. For large files (on the order of 1 million+ records), the SQL*Loader Direct Path load may be faster. Refer to *Oracle8i Utilities* for more information on SQL*Loader options.

Remember that the m1-m12 columns in the load table correspond directly to the MONTH_01 - MONTH_12 columns in Ledger_Stat.

For example, if your fiscal period runs from September through August, then MONTH_01 in Ledger_Stat contains values for September. Therefore, the m1 column in the load table also contains values for September because it is loaded directly into MONTH_01 in Ledger_Stat.

Also, the value in the YEAR_S column for each row in the load table should contain the same value as the corresponding row in Ledger_Stat. The YEAR_S value in Ledger_Stat is the calendar year of the first month of the fiscal period. For a load table row containing data from September, 1997 through August, 1998 in columns m1 through m12, respectively, the YEAR_S value should be 1997. See the *Oracle*

Performance Analyzer Reference Guide and *Oracle Financial Data Manager Administration* for additional information about fiscal year.

NULLs (blanks in the ASCII file) do not cause problems in the ONE_MONTH_AMT column or any of the MXX columns and, in fact, are recommended for unused amount columns. NULL values result in more compactly stored data in the load table and faster run times for the load procedure. NULL values are converted to zeroes during the load to Ledger_Stat.

You can load more than one data source into a single load table, using one SQL*Loader step or several. The Ledger_Stat load procedure can handle multiple data sources in the same load table. The primary advantage of loading from more than one load table is, for large Ledger_Stat loads (on the order of 500,00+ to millions of rows), the ability to load concurrently from multiple load tables.

Multi-Currency-Related Data Issues: ISO_CURRENCY_CD

For multi-currency support, FDM 4.5 includes the ISO_CURRENCY_CD column on the LEDGER_STAT table and as a component of its unique index. The column must also be added to any load table used for loading LEDGER_STAT. The FDM 4.5 scripts for creating a load table and the required views on the load table include this column. In addition, both the template SQL*Loader control file for the load table and the script for creating the LSL view on LEDGER_STAT include the ISO_CURRENCY_CD column. If you have upgraded from a previous version of OFSA, rerun these scripts to recreate your load tables and required views with the new definitions that include ISO_CURRENCY_CD.

Because ISO_CURRENCY_CD is a component of the unique index on LEDGER_STAT, the *upsert* logic of the Ledger Stat Load utility requires these values to be correct prior to attempting the load. The Ledger Stat Load procedure validates and updates ISO_CURRENCY_CD values directly in the load table prior to loading, in the following ways:

If multi-currency is enabled (OFSA_DB_INFO.MULTI_CURRENCY_ENABLED_FLG = 1), then the Ledger Stat Load procedure updates the ISO_CURRENCY_CD column in the load table as follows:

- For rows having a currency-basis financial element value (for example, balances, weighted rates) that contain an invalid value for ISO_CURRENCY_CD, the ISO_CURRENCY_CD column is set to the functional currency.
- For rows having a non-currency-basis financial element value (for example, statistics) that are not already set to the non-currency value for ISO_CURRENCY_CD in LEDGER_STAT (’002’), the ISO_CURRENCY_CD column is set to ’002’.

If multi-currency is not enabled (OFSA_DB_INFO.MULTI_CURRENCY_ENABLED_FLG = 0), the Ledger Stat Load procedure updates ISO_CURRENCY_CD to the functional currency for all rows where ISO_CURRENCY_CD is not already equal to the functional currency.

> **Note:** Only rows containing invalid or inappropriate values are updated. Avoid slowing down your load processes by populating the load table with correct ISO_CURRENCY_CD values before running the Ledger Stat Load utility.

Editing the Ledger_Stat Load Batch Parameter Table

Run the LS_LOAD_BATCH Data Verification ID (seeded by FDM). For each load table you want to load from, enter a row in the spreadsheet and fill in the runtime parameters.

Refer to the following table for a description of each of the parameters in Ledger_Stat Load Batch. The Load Month Column, First Load Month Column, and Last Load Month Column parameters refer to month columns rather than actual months.

The following table identifies the batch load parameters and the values, descriptions or usage, as appropriate, for each.

Parameter	Valid Description/Values/Usage
Process Flag (Y / N)	Y: Run a process for this batch record. N: Do not run a process for this batch record.
Processing Sequence	1-99 The processing order for batch records. The most efficient method, if you are running multiple processes, is to order the tables from largest to smallest. The ordering numbers do not need to be consecutive but they must be unique.
Load Table Name	The name of the table to load from.
One-Month-Only (Y / N)	Y: Load the month column designated by the “Load Month Column” parameter from the ONE_MONTH_AMT column in the load table. N: Load from the MONTH_XX (MXX) columns in the load table into the same-named columns in Ledger_Stat. The “First Load Month Column” and “Last Load Month Column” parameters must contain the month numbers indicating the range of MONTH_XX columns to be loaded.
Load Month Column (01-12)	The month number (01-12) for the month column to be loaded, when the “One-Month-Only” parameter ='Y'.

Parameter	Valid Description/Values/Usage	
First Load Month Column (01-12)	The month number (01-12) of the first column in the range of MONTH_XX columns to be loaded, when the "One-Month-Only" parameter = 'N'.	
Last Load Month Column (01-12)	The month number (01-12) of the last column in the range of MONTH_XX columns to be loaded, when the "One-Month-Only" parameter = 'N'.	
Update Mode (ADD / REPLACE)	ADD:	Adds (sums) input values to existing values in Ledger_Stat columns when updating existing Ledger_Stat rows.
	REPLACE:	Replaces existing values in Ledger_Stat columns with new input values when updating existing Ledger_Stat rows. This mode prevents you from accidentally double-loading your LEDGER_STAT table. Note that this parameter has nothing to do with the way Ledger_Stat rows are *upserted*. The only two methods for updating Ledger_Stat are: ▪ Updating the row if it already exists ▪ Inserting the row if it does not exist
Insert-Only (Y / N)	Y:	Bypasses the UPDATE portion of the "upsert" logic and inserts load_table rows directly into Ledger_Stat without checking for existing matching rows. Use this setting for faster execution when running first-time loads.
	N or blank:	Updates Ledger_Stat using the standard "upsert" logic, meaning that existing rows are updated and new rows are inserted.
Create Offsets (Y / N)	Y:	Create Offset records.
	N or blank:	Do not create Offset records.
READ-ONLY parameters, filled in by the load procedure for your information:		
Processing Start Date / Time	Filled in by the procedure with the start date / time of the current run.	
Processing End Date / Time	Filled in by the procedure with the completion date / time of the current run.	
Number of Rows Loaded	Filled in by the procedure with the number of Ledger_Stat rows updated or inserted.	

Parameter	Valid Description/Values/Usage
Message	Filled in by the procedure as “Successfully Completed” after a successful run, or with an error message in the event that the procedure detects an error in the batch record or traps a run-time error and terminates processing.

When you run the load procedure, the first record (the one with the lowest sequence number) in the Ledger_Stat Load Batch table that has a Processing Flag = Y is read; the Processing Flag is immediately set to N. The procedure checks the parameters in this row for correctness, completeness and consistency and then proceed to load Ledger_Stat from the specified load table. Once the load from this table is complete, the procedure reads the next row in sequence with a Processing Flag = Y. It continues in this loop until all batch / parameter records have been processed.

Invoking the Load Procedure

The are two possibilities for invoking the Load Procedure:

- running the seeded SQL ID RUN_LEDGER_LOAD from within Oracle Balance & Control
- running the procedure directly from SQL*Plus

To run from SQL*Plus enter the following command at the SQL> prompt:

```
SQL> execute ofsa_util.ledger_stat_load;
```

To run from Balance & Control, open the SQL ID RUN_LEDGER_LOAD from the OFSA ID Folder. Once the ID is open, click on the *Run* icon.

The best way to invoke the procedure is from a telnet SQL*Plus session so that client PC resources are not required. Invoking the procedure from Balance & Control or a client SQL*Plus session requires client PC resources (even though the procedure actually executes on the server).

Running the Synchronize Instrument Utility

If your load table includes new values for any of the leaf columns (values that have not been defined in Leaf Setup) run OFSA_UTIL.SYNCHRONIZE_INSTRUMENT. This utility posts default values in Leaf Setup for these new leaf values and adds the new leaf values to the orphan node of each Tree Rollup ID using that leaf column.

The leaves created in this way show *No description* in the description column in Leaf Setup. Go into Leaf Setup and complete the entries for these new values. See

Executing the SYNCHRONIZE_INSTRUMENT Procedure for more details on completing this task.

New values for FINANCIAL_ELEM_ID must be defined in Leaf Setup prior to loading or else rows containing these values are not loaded (see "Defining Financial Elements in Leaf Setup" for additional information).

Running Concurrent Loads with Multiple Load Tables

If you have large loads from multiple data sources, you may want to invoke multiple instances of the load procedure to concurrently load from multiple load tables into Ledger_Stat. To do this, set up the batch / parameter table as usual, with one row per load table.

To invoke the multiple procedures from SQL*Plus, open multiple SQL*Plus windows, or open multiple telnet windows, with each running SQL*Plus on the server. Invoking multiple procedures using the SQL ID requires multiple Balance & Control sessions.

Using either process, the first instance you invoke picks up the first row in the Ledger_Stat Load Batch table and begin loading Ledger_Stat from the load table specified in that batch / parameter record. The next procedure you invoke gets the next batch / parameter row, and so on. You could submit one procedure instance for each row in your batch / parameter table or you could have six rows representing six load tables, and invoke two procedures that would alternate in getting a row, processing it, getting another row, processing it and so forth, until all rows had been processed.

To maximize parallelization, order the batch / parameter rows from the largest load to smallest.

Undoing Ledger_Stat Load Updates

You can undo updates to Ledger_Stat by accessing the Process / Undo menu in Performance Analyzer. This undo procedure works, whether you have updated Ledger_Stat through Performance Analyzer or through the server-based Ledger_ Stat load procedure.

Invoking the undo function will zero-out the column in Ledger_Stat corresponding to the month(s) that you are undoing and will zero-out the rows having IDENTITY_ CODE values corresponding to the DATA_SOURCE that you are undoing.

Note the following qualifications to the undo process:

- If you are logged into Balance & Control from either a SQL ID or SQL*Plus when you run the load procedure, you must log out of Balance & Control and

log back in, in order for the record of your recent updates to become visible in the Process/Undo spreadsheet.

- If you used Update_Mode = ADD to load Ledger_Stat then, when you undo the load, you lose any values previously held in any existing rows for the months that you undo.

Using the Update Mode Parameter

The Update Mode parameter determines the manner in which MONTH columns are updated for existing rows in Ledger_Stat. The following two choices are available:

- Update Mode = ADD
- Update Mode = REPLACE

In most cases the MONTH column(s) you are loading contains zeros so you use Update Mode = REPLACE. This mode overwrites any existing values with the values you are loading. This feature prevents you from inadvertently double loading your Ledger_Stat and enables the load to be re-run, if necessary, without having to run the undo procedure.

Occasionally you may have rows in Ledger_Stat that already contain values in the MONTH column(s) corresponding to the month(s) you are loading. Use Update Mode = ADD only when you want the values you are loading to be added (summed) to the existing values in the existing Ledger_Stat rows that match the rows in your load table.

Using the Insert Only Parameter

This parameter, when set to Y(es) enables you to opt for a faster load when the data sources you are loading are being loaded for the first time or you are re-loading data sources after they have been completely deleted from Ledger_Stat.

Selecting the Insert Only = Y parameter bypasses the update portion of the upsert logic, and inserts all rows from the load table into Ledger_Stat. With this parameter selected there is no update function. This works because in these cases there are no existing rows in Ledger_Stat corresponding to the ones being loaded, so the update steps are not needed. However, if there are any matching rows, the load fails.

This mode is useful for first-time loads, or for clients who want to re-load their entire Ledger_Stat each period.

If you want to use the Insert Only = Y parameter for any load after the first-time load, you must either truncate the LEDGER_STAT table or delete from Ledger_Stat

all rows with IDENTITY_CODE values corresponding to the data source(s) you are going to load.

When you select the Insert Only = N parameter the load procedure first updates matching rows, then inserts new ones. This is the standard load parameter setting.

Using the Create Offsets Parameter

See Ledger_Stat load documentation in the *Performance Analyzer Reference Guide* for the procedure to create offsets.

Troubleshooting the Load Procedure

If a load procedure is only partially successful and loads some but not all of the rows from your load table, it is probably due to one of the following reasons:

- A data source has been duplicated among multiple load tables. A single data source value (column_name = ds in the load table) should not be present in multiple load tables being loaded concurrently. However, each load table may contain several distinct data source values.
- The process flag in the Ledger Stat Load Batch table is set to N for some or all jobs. Be aware that the process flag is reset to N after each run, even if the job is not successful. Prior to running any jobs, verify that this flag is set to Y.
- Some of the rows in the load table contain values for FINANCIAL_ELEM_ID that are undefined or incompletely defined.
- Some of the rows in the load table contain zero or null values for all of the columns being loaded. The view created against each load table filters out such rows. If you want to load rows containing all zeros for every month being loaded, you need to customize the lsldtbl.sql script to remove the WHERE clause from the <load_table_name>_V view and rerun the portion of the script that creates this view.

In either case, if you have selected Update Mode = REPLACE, then you can rerun the entire load after you have corrected the problem. There is no need to undo the load.

If you selected Update Mode = ADD, then you should collect the rows that were not loaded into a new load table and run a new load only for those rows. The reason is that if you selected Update Mode = ADD, then you would have to undo the load first, but this would also undo the portion of each value carried forward from previous loads.

Modify Balance Column Size

Balance columns store monetary values, such as a savings account balance, or a fee balance. FDM permits modification the Data Length, Data Scale and Data Precision for such columns with the **modify_balance_column_size** procedure.

> **Note:** Although FDM supports increasing the Data Length, Data Scale and Data Precision for Balance columns, this does not mean that the OFS processing engines output results to a greater precision. The precision for results generated by any of the OFS processing engines is independent of column definitions.

The modify_balance_column_size procedure alters the column definitions for all BALANCE columns in a specified category of tables registered within the FDM Metadata (with the exception of Services tables). FDM identifies such columns by querying from the OFSA_TAB_COLUMNS_V view where the FDM Data Type is BALANCE (ofsa_data_type_cd = 1) and the table_name is a member of the type of table specified in the procedure parameters. All tables that are within the designated category are altered.

For FDM Reserved columns, FDM requires that the column definition is the same across all tables on which the column exist. The column definitions are stored in OFSA_COLUMN_REQUIREMENTS. The **modify_balance_column_size** procedure updates the entries in OFSA_COLUMN_REQUIREMENTS to reflect any column size modifications. This keeps the column definitions in sync with the Table Classification requirements, so that all future registrations of objects with FDM Reserved columns new definition. It also ensures that all Balance columns on registered tables have the same definition.

> **Note:** Always use the modify_balance_column_size procedure to modify BALANCE column definitions. If you alter FDM Reserved column definitions manually, new Table Classification assignments fail because the column requirements in OFSA_COLUMN_ REQUIREMENTS do not match the new definitions.
>
> FDM does not support modification of the Data Length, Data Precision or Data Scale for non-Balance FDM Reserved columns.

Services Tables

The modify_balance_column_size procedure does not alter any BALANCE columns on Customer Householding Services tables. This is because FDM does not support expanded definitions for these columns on the Customer Householding Services tables. However, FDM does support expanded definitions for BALANCE columns not accessed by the Customer Householding processing. If you are using the Services tables for Profitability, Risk Manager or Transfer Pricing processing, then you must manually alter any of the columns required by these processing operations to match the new definitions in OFSA_COLUMN_REQUIREMENTS.

Executing the Procedure

The modify_balance_column_size procedure is part of the OFSA_UTIL package. To run the procedure, login to SQL*Plus as the FDM Schema Owner and execute the following command:

```
SQL> set serveroutput on
SQL> execute ofsa_util.modify_balance_column_size(p_table_type,p_
precision,p_scale);
```

where:

p_table_type is the category of table to be altered. There are 3 acceptable values for this parameter:

instrument - This category includes:

- Tables of Table Classification *Instrument* (table_classification_cd = 20) excluding Services tables
- OFSA_LEDGER_STAT_INSTRUMENT
- OFSA_LEDGER_STAT_RECON

ledger_stat - This category includes:

- LEDGER_STAT

rm_template - This category includes:

- OFSA_CONSOLIDATED_MASTER
- OFSA_EAR_LEAF_AVG
- OFSA_EAR_LEAF_DTL
- OFSA_EAR_TOTAL_AVG

- OFSA_EAR_TOTAL_DTL
- OFSA_IDT_CONSOLIDATED_DETAIL
- OFSA_IDT_RESULT_DETAIL
- OFSA_RESULT_MASTER

  The CUR_WARM column on OFSA_CONSOLIDATED_MASTER or OFSA_RESULT_MASTER is altered only when the column definition is expanded.

p_precision is the new Data Precision for the column

p_scale is the new Data Scale for the column

For example:

```
<SQL> execute ofsa_util.modify_balance_column_size('instrument',16, 2);
```

Or

```
<SQL> execute ofsa_util.modify_balance_column_size('ledger_stat',17, 4);
```

Limitations

If the Data Precision or Data Scale is decreased instead of expanded, the alter table statements fail where the BALANCE columns contain data. In this situation some of the alter statements may succeed while others fail. Oracle does not recommend that you ever decrease the Data Precision or Data Scale.

Review Log Information

After the procedure is complete, review log entries in the OFSA_STP table where TASKNAME = modify_balance_column_size. These log entries provide information on whether or not the procedure was successful.

Recompiling Packages, Procedures, and Java Classes

The FDM database includes PL/SQL packages, procedures, and Java Classes. All of these are loaded into the database by either the FDM database creation process or database upgrade process. These objects are a required component of the FDM database definition.

It is possible for one or more of these objects to become INVALID. This may occur during a database import, or because an object reference by the package, procedure or java class no longer exists in the database. You or your users may receive Oracle errors indicating this situation with such errors as:

- ORA-04068 Existing state of packages has been discarded
- Unable to resolve Java Class

To identify if this is the case, run the following query in SQL*Plus as the FDM Schema Owner:

```
select object_name, object_type, object_status
from user_objects
where status = 'INVALID';
```

Included in the utilities directory with the FDM database scripts is a script to refresh these objects, in the event that one or more of these objects becomes invalid. The default location for this refresh script is the **OFSA_INSTALL/dbs/<OFSA release>/utilities/stp** subdirectory of your OFSA installation directory. OFSA_INSTALL is the convention used to indicate where the OFSA software is installed in your directory structure.

Financial Data Manager Packages

The Financial Data Manager packages consist of all packages, procedures and java classes required by the OFS applications, excluding those required exclusively by Oracle Market Manager.

To run the script, go to this directory location and login to SQL*Plus as the FDM Schema Owner. Then type the following:

```
<SQL> @fdm_packages.sql
```

The fdm_packages.sql script prompts for the password of the FDM Schema Owner. This password is necessary for creating the java classes in the database.

This script creates the following log files in the **OFSA_INSTALL/dbs/<OFSA release>/log** directory:

- fdm_packages.log
- pass_jar_status.log
- rtm_jar.log

Review these logs for any errors.

Market Manager Packages

The Market Manager packages consist of packages and stored procedures required by Market Manager, including some that are also required by FDM Administration.

To run the script, go to this directory location and login to SQL*Plus as the FDM Schema Owner. Then type the following:

```
<SQL> @mm_packages.sql
```

This script creates the following log files in the **OFSA_INSTALL/dbs/<OFSA release>/log** directory:

- mm_packages.log

Review this log for any errors.

Recompiling Views and Triggers

The FDM database includes both views and triggers. These objects are loaded into the database by either the FDM database creation process or database upgrade process. These objects are a required component of the FDM database definition.

It is possible for one or more of these objects to become INVALID. This may occur during a database import, or because an object reference by the package, procedure or java class no longer exists in the database. You or your users may receive Oracle errors indicating this situation with such errors as:

- ORA-04063: view has errors
- Errors or incorrect behavior during insert, update or deleting on view objects

To identify if this is the case, run the following query in SQL*Plus as the FDM Schema Owner:

```
select object_name, object_type, object_status
from all_objects
where owner = fdm_schema_owner
and status = 'INVALID';
```

Included in the utilities directory with the FDM database scripts is a script to refresh these objects, in the event that one or more of these objects becomes invalid. The default location for this refresh script is the **OFSA_INSTALL/dbs/<OFSA release>/utilities/views** subdirectory of your OFSA installation directory. OFSA_INSTALL is the convention used to indicate where the OFSA software is installed in your directory structure.

Financial Data Manager Views

The Financial Data Manager views consist of all views required by the OFS applications, excluding those required exclusively by Market Manager.

To run the script, go to this directory location and login to SQL*Plus as the FDM Schema Owner. Then type the following:

```
<SQL> spool fdm_views.log
<SQL> @fdm_views.sql
```

Review the log file for any errors. Because the fdm_views.sql script drops and re-creates views, the privileges for these objects need to be regranted. After the script completes successfully, run the FDM Grant All procedure to regrant these privileges.

Market Manager Views

The Market Manager views consists of views required by Market Manager.

To run the script, go to this directory location and login to SQL*Plus as the FDM Schema Owner. Then type the following:

```
<SQL> spool mm_views.log
<SQL> @mm_views.sql
```

Review the log file for any errors. After the script completes successfully, run the Grant All procedure. Because the mm_views.sql script drops and re-creates views, the privileges for these objects need to be regranted.

Instrument Synchronization

During the period-ending load cycle, data is loaded into Client Data Objects such as Instrument tables and the LEDGER_STAT table. During this load process, it is possible for new, unidentified Leaf and Code values to be loaded into these tables. The Instrument Synchronization procedure identifies these new Leaf and Code values and inserts default description entries for them into the appropriate FDM tables. The procedure performs both of these synchronizations simultaneously. It also calculates statistics for the Undo function in Performance Analyzer.

FDM requires that all Leaf and Code values have a corresponding description. This is required for any OFSA reporting operation to return the correct results. It also ensures that Tree Rollup IDs work properly within the OFS applications.

Note: The SYNCHRONIZE_INSTRUMENT utility is a stored procedure that is run by the administrator. It is recommended that you include this procedure to run after the data load as a regular part of your period-ending processing cycle.

The following topics are covered in this section:

- Tables Requiring Synchronization
- Leaf Synchronization
- Code Synchronization
- Performance Analyzer Undo Statistics
- Executing the Synchronize Instrument Procedure
- Exception Messages

Tables Requiring Synchronization

Leaf and Code value synchronization is required only for Instrument and LEDGER_STAT tables. Instrument tables are defined as all tables with the *Instrument* Table Classification (table_classification_cd = 20) on which all of the defined Leaf Columns exist.

Leaf Synchronization

The SYNCHRONIZE_INSTRUMENT procedure synchronizes the Leaf Setup tables and the Rollup table with Ledger_Stat and instrument tables, using default values for leaf descriptions and other leaf information columns. You can then add the correct data to the new leaf values in Leaf Setup.

The procedure performs the following functions:

- Checks the specified table (Ledger_Stat or instrument) for new leaf values in each of that table's leaf columns and adds the new leaf values to the OFSA Leaf Description (OFSA_LEAF_DESC) table.
- Adds the new leaf values in the OFSA_LEAF_DESC table to the corresponding detail leaf tables.
- Adds the new leaf values, as orphan leaves, to the corresponding Tree Rollup IDs (OFSA_IDT_ROLLUP).

When new leaf values are added to the leaf setup tables these leaves include *No Description* in the DESCRIPTION column and contain default values in other leaf information columns. After the SYNCHRONIZE_INSTRUMENT utility run is completed the you should look for any new leaf values using the OFSA Leaf Setup menu and enter the correct descriptions and other leaf information.

You should also look at the orphan node of each Tree Rollup ID for new leaf values and move these new values to the appropriate branch in the rollup.

Codes Synchronization

The SYNCHRONIZE_INSTRUMENT procedure identifies code values in Instrument and LEDGER_STAT tables for which a corresponding description does not exist and inserts a default description into the appropriate Code Description object. This applies only to *CODE* columns categorized as User Editable or User Defined. *CODE* columns for which FDM reserves all of the values are not updated by this procedure. The procedure displays a warning message for any unidentified values in *CODE* columns where FDM reserves the entire range.

For each *CODE* column (ofsa_data_type_cd = 3) on the specified object, the SYNCHRONIZE_INSTRUMENT procedure queries from OFSA_DESCRIPTION_TABLES to identify the object storing the corresponding descriptions. If the resulting object is a User Editable or User Defined Code Description object, then the procedure inserts a default description for any code values for which a description record does not already exist. If the resulting object is an FDM Reserved Code Description object, then the procedure outputs a warning message indicating how many invalid code values exist in the specified Instrument or LEDGER_STAT table.

Refer to Chapter 16, "FDM Object Management" for more information about objects and reserved seeded data ranges.

For example, if you are synchronizing the DEPOSITS table, the procedure queries all of the *CODE* columns on this table. An example of an FDM Reserved *CODE* column is ACCRUAL_BASIS_CD. If the procedure finds any code values in this column that are not present in the corresponding Code Description object (OFSA_ACCRUAL_BASIS_DSC), it outputs an error message indicating the number of invalid values present.

FDM Reserved Code Description objects are identified by the following SQL statement:

```
select table_name from ofsa_table_class_assignment
where table_classification_cd = 197;
```

An example of a User Editable *CODE* column is SIC_CD. If the procedure finds any code values in SIC_CD in the DEPOSITS table that do not have a description in OFSA_SIC_DSC, it creates a default description No Description for each value. It is then up to the users to update these descriptions as appropriate.

User Editable Code Description objects are identified by the following SQL statement:

```
select table_name from ofsa_table_class_assignment
where table_classification_cd = 198;
```

User Defined Code Description objects are object not created by FDM. You create and register these objects for user-defined *CODE* columns. User Defined Code Description objects are identified by the following SQL statement:

```
select table_name from ofsa_table_class_assignment
where table_classification_cd = 298;
```

Performance Analyzer Undo Statistics

The SYNCHRONIZE_INSTRUMENT procedure also calculates statistics used by Performance Analyzer for the Undo function. It calculates the statistics automatically during the synchronization based upon the AS_OF_DATE specified.

Executing the SYNCHRONIZE_INSTRUMENT Procedure

You can execute this procedure from either SQL*Plus or from within a PL/SQL block. To run the procedure, login to SQL*Plus as the FDM Schema Owner. The procedure requires 2 parameters - table name to be synchronized and the As of Date. Identify the table name parameter by enclosing it in single quotes and uppercase, as shown in the following two examples.

The syntax for calling the procedure is:

```
ofsa_util.synchronize_instrument('TABLE_NAME', AS_OF_DATE)
```

where table_name is either:

- The name of an Instrument table
- LEDGER_STAT

Specify the AS_OF_DATE in the appropriate date format, based upon your NLS_DATE_FORMAT parameter for the database.

An example of running the stored procedure from SQL*Plus for the DEPOSITS table follows:

```
SQL> set serveroutput on

SQL> execute ofsa_util.synchronize_instrument('DEPOSITS', '05/31/2000');
```

Note: The AS_OF_DATE parameter is not required. However, if you do not specify it, the procedure calculates Performance Analyzer Undo statistics for all distinct AS_OF_DATE values in the specified table. This is unnecessary and may result in the procedure requiring substantially more time to complete. Oracle recommends specifying the AS_OF_DATE parameter to improve performance of the SYNCHRONIZE_INSTRUMENT procedure.

To execute the stored procedure from within a PL/SQL block or procedure see the example that follows. Call the procedure as often as required to synchronize all of your instrument tables. The appropriate table name and AS_OF_DATE is enclosed in single quotes.

```
BEGIN
.
.
ofsa_util.synchronize_instrument('LEDGER_STAT', '05/31/2000');
.
.
END;
```

Exception Messages

The SYNCHRONIZE_INSTRUMENT procedure may cause two exceptions to appear. The text and explanation for each of these exceptions follows. If you call the procedure from a PL/SQL block you may want to handle them so that your program can proceed.

Exception 1: Invalid table

The exception message reads:

```
ORA-20001 Invalid table name passed as a parameter.
```

where table_name is the name of the instrument table passed as a parameter to the procedure.

This exception occurs when the value supplied for the table_name parameter is not registered in the FDM Metadata.

Exception 2: Table is not an Instrument or LEDGER_STAT table

The exception message reads:

```
ORA-20002 Cannot process: table_name is not an OFSA Instrument or Ledger
type table having all leaf columns.
```

This exception occurs when the table_name parameter is not designated as an Instrument table or LEDGER_STAT table in the FDM Metadata. The procedure identified such tables based upon the Table Classification (Instrument or Ledger Stat). Refer to theTables Requiring Synchronization section for more information.

Exception 3: Leaf Desc has invalid seeded FINANCIAL_ELEM_ID values

The exception message reads:

```
ORA-20003 Cannot process: Seeded range in LEAF_DESC has too many FINANCIAL_
ELEM_ID values.
```

This error occurs when user-defined leaf values are found in the OFSA_LEAF_ DESC table within the FDM Reserved seeded data range. The FDM seeded data range for OFSA_LEAF_DESC is WHERE LEAF_NUM_ID=0 and LEAF_ NODE<10000. If more records are found in this range than the seeded count for FDM version, the Synchronize Instrument procedure displays the error message and terminates. Delete any user-defined Financial Element leaf values within the FDM seeded data range in order to resolve this problem.

Exception 4: Table has invalid seeded FINANCIAL_ELEM_ID values

The exception message reads:

```
ORA-20004 Cannot process: table_name has new FINANCIAL_ELEM_ID values that
are within seeded range (less than 10000).
```

This error occurs when user-defined leaf values are found in the OFSA_LEAF_ DESC table within the FDM Reserved seeded data range. The FDM seeded data range for OFSA_LEAF_DESC is WHERE LEAF_NUM_ID=0 and LEAF_ NODE<10000. If more records are found in this range than the seeded count for FDM version, the Synchronize Instrument procedure displays the error message and terminates. Delete any user-defined Financial Element leaf values within the FDM seeded data range in order to resolve this problem.

Reporting Utilities

Reporting utilities include any scripts or control files used to facilitate reporting for the OFSA Reporting Data Mart. Two such utilities are provided with OFSA 4.5 for

the purpose of special reporting functions using the OFSA FDM Business Area standard reports. The following control files are included as Reporting Utilities with FDM 4.5:

- lsrecon.ctl
- lsinstr.ctl

The purpose of this chapter is to introduce these control files and provide an overview of the OFSA Reporting functions that use them. However, for detailed information about how to run the OFSA Reconciliation or Ledger Stat Instrument reports, refer to the *Oracle Financial Data Manager Reporting Administration Guide*.

The following topics are included in this section:

- Overview of FDM Utilities -Reporting files
- Instructions for customizing the *lsrecon.ctl* and *lsinstr.ctl*

Overview

The FDM Business Area contains two reports that require the use of SQL*Loader control files. These reports are:

- OFSA Reconciliation Report
- OFSA Ledger Stat Instrument Report

The OFSA Reconciliation Report shows differences between instrument tables and the LEDGER_STAT table. While this report is run from the FDM Business Area in Discoverer, some additional data population is required if you intend to actually reconcile the results by posting values to LEDGER_STAT. In this situation, you use SQL*Loader to load the posted values into the OFSA_LEDGER_STAT_RECON table with lsrecon.ctl as the control file.

The OFSA Ledger Stat Instrument Report is used to facilitate running Oracle Transfer Pricing against LEDGER_STAT data. Because transfer pricing processing does not run against a LEDGER_STAT table structure, the OFSA Ledger Stat Instrument report is used to transfer LEDGER_STAT data to a table with an appropriate table structure so that transfer pricing can occur. The lsinstr.ctl control file is used to load results from the Ledger Stat Instrument Report into a table with the correct structure for transfer pricing.

Customizing the Control Files

The lsrecon.ctl and lsinstr.ctl are template SQL*Loader control files provided with OFSA 4.5 for posting reconciliation results to the LEDGER_STAT table. These files are located in the **OFSA_INSTALL/dbs/<OFSA release>/utilities/reporting** directory. OFSA_INSTALL is the convention used to indicate where the OFSA software is installed in your directory structure. On the client side, these files are located in the **<Oracle Home>\<OFSA Release>\disco31** directory.

These files require customization in order to be used to post reconciliation results to LEDGER_STAT. Instructions for editing these control files are included. However, for detailed information and instructions on the OFSA Reconciliation Report and OFSA Ledger Stat Instrument Report processes, refer to the *Oracle Financial Data Manager Reporting Administration Guide*.

Make a copy of the control file that you are using (either lsrecon.ctl or lsinstr.ctl) for editing, then follow these steps to customize the file.

1. Modify the INFILE name to include the full path name and file name of the CSV data file from which you want to load.
2. Add a line for each user-defined leaf column in the list of columns at the bottom of the file.
3. Verify that all columns designated as NOT NULL are included in the file for loading.
4. Verify that the column order is the same between the control file and the Discoverer table columns exported to the .CSV file.
5. Call SQL Loader and run the process. This loads data from the CSV file to the table specified in the control file.

22

Sending Databases to Oracle Support Services

This chapter provides instructions for sending copies of your Oracle Financial Data Manager (FDM) database to Oracle Support Services for assistance.

Requirements of Oracle Support Services

In some situations, Oracle Support Services may request a copy of your FDM database in order to investigate an issue. The instructions described in this chapter provide a step-by-step procedure for creating the files required by Oracle Support Services in the appropriate format.

> **Note:** Oracle Support Services may require that you complete only a subset of the instructions described below for any particular situation. The instructions provided below are a complete set that may be required only in certain circumstances. Whenever you provide it a database, Oracle Support Services will inform you of the specific steps required for its investigation.

Provide the following files for Oracle Support Services:

Files	Comments/Instructions
A full Export of your database to tape	Oracle recommends that you export all your non-instrument tables to one tape and spread your instrument tables to multiple tapes.
A full Export of your database to tape	Use the following parameters: FULL = Y, ROWS = N.
An soft (electronic) copy of your init.ora file & config.ora.file	
A description of what is on each medium and how it was put on the medium	
A soft (electronic) copy of the Export Logs and the parameters of par files used to create the exports	
A soft (electronic) copy of the control file in text format	To obtain a copy of the control file in text format: 1. Log in to svrmgrl. 2. Type the following: `alter database backup controlfile to trace;` 3. Exit svrmgrl. 4. Change directory to the location specified by the init.ora parameter user_dump_dest Usually the latest trace file is the text version of the control file.

Files	Comments/Instructions
A soft (electronic) copy of the logging file produced by running the following script.	This script and support scripts are located on MetaLink. Perform the following steps: **1.** Login to SQL*Plus as SYSTEM. **2.** Start the first script by typing at the SQL> prompt and passing the name of the logging file as a parameter. The syntax is: `SQL> @get_info filename.log` For example: @get_info database_info.log **3.** Exit SQL*Plus.

A

Functional Currencies

Acceptable Values

The following table lists the Functional Currency values acceptable for the FDM database creation and database upgrade processes:

ISO Currency Code	Currency Name
ADP	Andorran Peseta
AED	United Arab Emirates Dirham
AFA	Afghanistan Afghani
ALL	Albanian Lek
AMD	Armenia Dram
ANG	Netherlands Antillian Guilder
AOK	Angolan Kwanza
ARS	Argentine Peso
ATS	Austrian Schilling
AUD	Australian Dollar
AWG	Aruban Florin
AZS	Azerbaijan Manat
BBD	Barbados Dollar
BDT	Bangladeshi Taka
BEF	Belgian Franc

ISO Currency Code	Currency Name
BES	Belarus Rouble
BGL	Bulgarian Lev
BHD	Bahraini Dinar
BIF	Burundi Franc
BMD	Bermudian Dollar
BND	Brunei Dollar
BOP	Bolivian Boliviano
BRL	Brazil Real
BSD	Bahamian Dollar
BTN	Bhutan Ngultrum
BUK	Burma Kyat
BWP	Botswanian Pula
BZD	Belize Dollar
CAD	Canadian Dollar
CHF	Swiss Franc
CLF	Chilean Unidades de Fomento
CLP	Chilean Peso
CNY	Yuan (Chinese) Renminbi
COP	Colombian Peso
CPF	French Pacific Island Franc
CRC	Costa Rican Colon
CUP	Cuban Peso
CVE	Cape Verde Escudo
CYP	Cyprus Pound
CZK	Czech Koruna
DDM	East German Mark (DDR)
DEM	German Deutsche Mark
DJF	Djibouti Franc

ISO Currency Code	Currency Name
DKK	Danish Krone
DOP	Dominican Peso
DZD	Algerian Dinar
EEK	Estonian Kroon
EGP	Egyptian Pound
ESP	Spanish Peseta
ESS	Ecuadoran Sucre
ETB	Ethiopian Birr
EUR	Euro (European EMU)
FIM	Finnish Markka
FJD	Fiji Dollar
FKP	Falkland Islands Pound
FRF	French Franc
GBP	British Pound
GEL	Georgian Lari
GHC	Ghanaian Cedi
GIP	Gibraltar Pound
GMD	Gambian Dalasi
GNS	Guinea Franc
GRD	Greek Drachma
GTQ	Guatemalan Quetzal
GWP	Guinea-Bissau Peso
GYD	Guyanan Dollar
HKD	Hong Kong Dollar
HNL	Honduran Lempira
HRK	Croatian Kuna
HTG	Haitian Gourde
HUF	Hungarian Forint

ISO Currency Code	Currency Name
IDR	Indonesian Rupiah
IEP	Irish Punt
INR	Indian Rupee
IQD	Iraqi Dinar
IRR	Iranian Rial
ISK	Iceland Krona
ISS	Israeli Shekel
ITL	Italian Lira
JMD	Jamaican Dollar
JOD	Jordanian Dinar
JPY	Japanese Yen
KES	Kenyan Schilling
KHR	Kampuchean (Cambodian) Riel
KMF	Comoros Franc
KPW	North Korean Won
KRW	(South) Korean Won
KTS	Kazakhstan Tenge
KWD	Kuwaiti Dinar
KYD	Cayman Islands Dollar
KYS	Kyrgyzstan Som
LAK	Lao Kip
LBP	Lebanese Pound
LKR	Sri Lanka Rupee
LRD	Liberian Dollar
LSM	Lesotho Loti
LTT	Lithuanian Lit
LUF	Luxembourg Franc
LVR	Latvian Lat

ISO Currency Code	Currency Name
LYD	Libyan Dinar
MAD	Moroccan Dirham
MGF	Malagasy Franc
MMK	Myanmar (Burma) Kyat
MNT	Mongolian Tugrik
MOP	Macau Pataca
MRO	Mauritanian Ouguiya
MTL	Maltese Lira
MUR	Mauritius Rupee
MVR	Maldive Rufiyaa
MVS	Moldova Lei
MWK	Malawi Kwacha
MXN	Mexican Peso
MYR	Malaysian Ringgit
MZM	Mozambique Metical
NGN	Nigerian Naira
NIC	Nicaraguan Cordoba
NLG	Dutch Guilder
NOK	Norwegian Kroner
NPR	Nepalese Rupee
NZD	New Zealand Dollar
OMR	Omani Rial
PAB	Panamanian Balboa
PGK	Papua New Guinea Kina
PHP	Philippine Peso
PKR	Pakistan Rupee
PLN	Polish Zloty
PSS	Peruvian New Sol

ISO Currency Code	Currency Name
PTE	Portuguese Escudo
PYG	Paraguay Guarani
QAR	Qatari Rial
ROL	Romanian Leu
RUR	Russian Rouble
RWS	Rwanda Franc
SAR	Saudi Arabian Riyal
SBD	Solomon Islands Dollar
SCR	Seychelles Rupee
SDD	Sudanese Pound
SEK	Swedish Krona
SGD	Singapore Dollar
SHP	St. Helena Pound
SIT	Slovenia Tolar
SKK	Slovakia Koruna
SLL	Sierra Leone Leone
SOS	Somali Schilling
SRG	Suriname Guilder
STD	Sao Tome and Principe Dobra
SVC	El Salvador Colon
SYP	Syrian Pound
SZL	Swaziland Lilangeni
THB	Thai Baht
TJS	Tajikistan Ruble
TMS	Turkmenistan Manat
TND	Tunisian Dinar
TOP	Tongan Pa'anga
TPE	East Timor Escudo

ISO Currency Code	Currency Name
TRL	Turkish Lira
TTD	Trinidad and Tobago Dollar
TWD	Taiwan Dollar
TZS	Tanzanian Schilling
UAK	Ukraine Hryvna
UGS	Uganda Shilling
USD	US Dollar
UYP	Uruguayan Peso
UZS	Uzbekistan Sum
VEB	Venezuelan Bolivar
VND	Vietnamese Dong
VUV	Vanuatu Vatu
WST	Samoan Tala
XEU	European Currency Unit
YDD	Yemeni Dinar
YER	Yemeni Rial
YUD	New Yugoslavia Dinar
ZAR	South African Rand
ZMK	Zambian Kwacha
ZRZ	Zaire Zaire
ZWD	Zimbabwe Dollar

Glossary

A glossary is not provided with the *Oracle Financial Services Installation and Configuration Guide.*

Index

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

W