Oracle® Database

System Administration Guide

10*g* Release 2 (10.2) for IBM z/OS (OS/390)

B25398-01

April 2006

Oracle Database System Administration Guide, 10*g* Release 2 (10.2) for IBM z/OS (OS/390)

B25398-01

Primary Authors: Apolina Das, Sanjay Sharma

Contributing Author: Server Technologies Porting

Contents

11 Oracle Access Manager for CICS TS

Preface

This guide explains how to perform administrative tasks for Oracle Database 10*g* release 2 (10.2) for IBM z/OS.

Audience

This guide is intended for anyone responsible for performing tasks such as:

- Administering Oracle Database 10*g* release 2 (10.2) for IBM z/OS (OS/390)
- Maintaining z/OS data sets and Oracle system files
- Managing backups, recovery, memory, and storage on z/OS
- Diagnosing and reporting errors
- Issuing OSDI subsystem commands

This guide provides information only for Oracle products and their interactions with z/OS. A thorough understanding of the fundamentals of z/OS is necessary before attempting to use this software.

Documentation Accessibility

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

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

Accessibility of Code Examples in Documentation

Screen readers may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, some screen readers may not always read a line of text that consists solely of a bracket or brace.

Accessibility of Links to External Web Sites in Documentation

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

TTY Access to Oracle Support Services

Oracle provides dedicated Text Telephone (TTY) access to Oracle Support Services within the United States of America 24 hours a day, seven days a week. For TTY support, call 800.446.2398.

Product Name

The complete name for the product described in this book is Oracle Database 10*g* release 2 (10.2) for IBM z/OS (OS/390). To maintain readability and conciseness in this document, the product is referred to as Oracle Database for z/OS, and the platform is referred to as z/OS.

Command Syntax

UNIX command syntax appears in `monospace` font. The dollar character ($), number sign (#), or percent character (%) are UNIX command prompts. Do not enter them as part of the command. The following command syntax conventions are used in this guide:

Convention	Description
backslash \	A backslash is the UNIX command continuation character. It is used in command examples that are too long to fit on a single line. Enter the command as displayed (with a backslash) or enter it on a single line without a backslash: `dd if=/dev/rdsk/c0t1d0s6 of=/dev/rst0 bs=10b count=10000`
braces { }	Braces indicate required items: `.DEFINE {macro1}`
brackets [ ]	Brackets indicate optional items: `cvtcrt` *`termname`* `[`*`outfile`*`]`
ellipses ...	Ellipses indicate an arbitrary number of similar items: `CHKVAL fieldname value1 value2 ... valueN`
italics	Italic type indicates a variable. Substitute a value for the variable: *`library_name`*
vertical line |	A vertical line indicates a choice within braces or brackets: `FILE` *`filesize`* `[K|M]`

Accessing Documentation

The documentation for this release includes platform-specific documentation and generic product documentation.

Platform-Specific Documentation

Platform-specific documentation includes information about installing and using Oracle products on particular platforms. The platform-specific documentation for this product is available in both Adobe portable document format (PDF) and HTML format on the product disc. To access the platform-specific documentation on disc:

1. Use a Web browser to open the `welcome.htm` file in the top-level directory of the disc.
2. For DVD only, select the appropriate product link.
3. Select the **Documentation** tab.

If you prefer paper documentation, then open and print the PDF files.

Product Documentation

Product documentation includes information about configuring, using, or administering Oracle products on any platform. The product documentation for Oracle Database 10*g* products is available in both HTML and PDF formats in the following locations:

- In the doc subdirectory on the Oracle Database 10*g* DVD

  To access the documentation from the DVD, use a Web browser to view the `welcome.htm` file in the top-level directory on the disc, then select the Oracle Database 10*g* Documentation Library link.

- Online on the Oracle Technology Network (OTN) Web site:

  `http://www.oracle.com/technology/documentation/index.html`

Related Documentation

The platform-specific documentation for Oracle Database 10*g* products includes the following manuals:

- Oracle Database
  - *Oracle Database Release Notes for IBM z/OS (OS/390)*
  - *Oracle Database How To Get Started for IBM z/OS (OS/390)*
  - *Oracle Database Installation Guide for IBM z/OS (OS/390)*
  - *Oracle Database User's Guide for IBM z/OS (OS/390)*
  - *Oracle Database Messages Guide for IBM z/OS (OS/390)*
  - *Oracle Database System Administration Guide for IBM z/OS (OS/390)*
- Other Documentation

  In addition to the platform-specific documents, the following product-specific documents are referenced in this guide.

  Oracle Books

  - *Oracle Database Administrator's Guide*
  - *Oracle Database Advanced Security Administrator's Guide*
  - *Oracle Database Application Developer's Guide - Fundamentals*
  - *Oracle Database Application Developer's Guide - Large Objects*
  - *Oracle Database Backup and Recovery Book Set*
  - *Oracle Database Concepts*
  - *Oracle Database Data Guard Concepts and Administration*

– *Oracle Database Globalization Support Guide*
– *Oracle Database Performance Tuning*
– *Oracle Database Real Application Clusters Book Set*
– *Oracle Database Recovery Manager Book Set*
– *Oracle Database Reference*
– *Oracle Database SQL Reference*
– *Oracle Database Upgrade Guide*
– *Oracle Database Utilities*
– *Oracle Enterprise Manager Advanced Configuration*
– *Oracle Enterprise Manager Book Set*
– *Oracle Net Services Book Set*

IBM Books

– *IMS V8 Install. Vol. 2: System Definition and Tailoring* (GC27-1298)
– *z/OS V1R3.0 DFSMS Access Method Services for Catalogs* (SC26-7394)
– *z/OS V1R3.0 DFSMS: Using Data Sets* (SC26-7410)
– *z/OS V1R4.0 MVS Initialization and Tuning Reference* (SA22-7592)
– *z/OS V1R4.0 MVS Planning: Workload Management* (SA22-7602)
– *z/OS V1R4.0 MVS Setting Up a Sysplex* (SA22-7625)
– *z/OS V1R4.0 UNIX System Services Planning* (GA22-7800)

IBM documents are referred in a shorter form throughout this document. For example, *z/OS V1R4.0 MVS Initialization and Tuning Reference* (SA22-7592) is referred to as *MVS Initialization and Tuning Reference.*

Refer to *Oracle Database Release Notes for IBM z/OS (OS/390)* for important information that was not available when this book was released. The release notes for Oracle Database 10*g* are updated regularly. You can get the most recent version from Oracle Technology Network at

`http://www.oracle.com/technology/documentation/index.html`

Typographic Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

1 Introduction to OSDI Architecture

OSDI (Operating System Dependent Interface) is a platform architecture for Oracle products running on z/OS. This chapter provides a broad overview of OSDI and introduces concepts and terminology used throughout this book.

The following topics are included:

- Overview
- Subsystem
- Services
- Connections
- Security
- Database Service
- Network Service

Overview

OSDI consists of several well-defined components. The first of these components is a common management layer that is shared by all Oracle products that are implemented under OSDI. Below this, separate components are provided for each Oracle product or type of product. OSDI currently has two product implementation components, one for the Oracle RDBMS and one for Oracle Net.

Subsystem

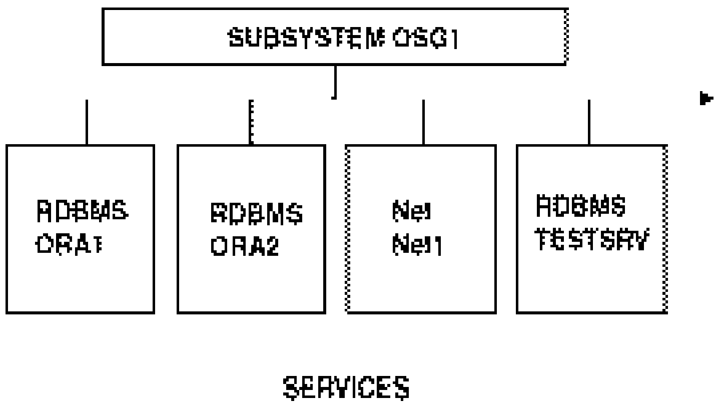
The common management layer of OSDI is implemented as a formal z/OS subsystem. It is not associated with any single Oracle product or product instance.

One OSDI subsystem can support multiple Oracle database instances and multiple Oracle Net services. The term **service group** refers to a collection of database instances and network services that are managed by an OSDI subsystem.

The OSDI subsystem has no associated address space. When the subsystem is initialized (normally during z/OS IPL), its program code is loaded into system common storage and a few data structures are built, but no address spaces are started. Subsequent interaction with the subsystem is accomplished via z/OS system operator commands and via programmatic requests issued over the subsystem interface (SSI). No address spaces are created until Oracle products are started.

The OSDI subsystem is what z/OS calls a dynamic subsystem. This means that it does not have to be initialized at system IPL: it can be initialized at any time using the SETSSI ADD system operator command. Once initialized, the OSDI subsystem is present in the z/OS system until the next IPL. Because the OSDI subsystem is a dynamic subsystem, it can be deactivated (SETSSI DEACTIVATE) and activated (SETSSI ACTIVATE) at will. When a subsystem is deactivated, no commands can be issued to it, no new Oracle product address spaces can be started, and no new client connections can be made to product instances currently running. Currently-running product instances and any existing client connections are not disturbed.

Figure 1–1 OSDI Architecture



Services

The primary role of the subsystem is to manage services. A service is a named, configured instance of an Oracle product. Each Oracle database instance and each network service that you run under OSDI will be a separate service. One subsystem can manage as many services as you desire. It is not necessary to create multiple OSDI subsystems in order to run multiple database instances or to run multiple network services.

Services must be defined to the subsystem. This is done with an OSDI DEFINE command. While OSDI commands can be issued from a z/OS system console (or similar interfaces such as SDSF or Netview), service definition commands are fairly lengthy and change infrequently. A significant advantage is gained by placing service definition commands in the parameter file that is read during OSDI subsystem initialization. Among the items that are specified when a service is defined are the user-selected service name, the type of Oracle product, and the name of the JCL procedure that is invoked to start an address space for the service. After a service is defined, the definition persists for the life of the IPL. The service cannot be deleted or renamed, but it is possible to change certain parts of the service definition using an OSDI ALTER command.

Defining or altering services manipulates only data structures that belong to the subsystem. These actions do not cause the execution of any Oracle products. In order to run a product that is configured as a service, an OSDI START command must be issued. The OSDI START command is similar to the z/OS command of the same name: it causes a specified service to begin execution in a z/OS address space using the JCL procedure specified in the service definition. You must use the OSDI START command (not the z/OS command) to start services. OSDI-managed services cannot be executed as z/OS batch jobs or as independent started tasks or STCs.

After a service is started, its behavior and operation depend on the Oracle product and how that product is configured. In general, the subsystem keeps track of the operating state of each service. The primary operating states are inactive, starting, active, and stopping. A service is normally in starting and stopping states only briefly, from the time it is started until initialization is complete (the service becomes active) and from the time termination begins until it actually ends (the service becomes inactive). Only services that are active are available for applications to use. You can display the state of a service using the OSDI DISPLAY command.

Connections

The subsystem is responsible for processing OSDI-specific commands and for managing the definition and operation of services. It also is responsible for managing connections between z/OS address spaces, specifically between clients and services. In this context, the term **client** differs from the usual Oracle usage of the term. A client is any z/OS address space that issues requests to an OSDI-managed service via z/OS cross-memory services. This includes applications that you normally think of as clients, such as an Oracle application or utility that connects to a local z/OS Oracle database instance. It also includes applications that are less obvious: when the Oracle network service connects an inbound remote client to a z/OS Oracle database, the network service address space is a client of the database service in OSDI terms.

OSDI calls this process of connecting a client address space to a service address space a **bind**. Bind processing is internal to Oracle product operation and is not directly visible to applications or users. The bind request uses the z/OS subsystem interface (IEFSSREQ macro), which is why deactivating a subsystem makes its services inaccessible to new client connections, or in other words, to new binds. OSDI has two types of bind requests. One type is used only by Oracle products that have special requirements for managing connections to services, including Oracle Net and the Oracle Access Manager products. The other type of bind is a normal application bind used by customer applications and by Oracle tools and utilities running in TSO, batch, and UNIX System Services address spaces. While bind is not an external mechanism, security considerations that are in bind processing are both visible and important. Security considerations are described in the following section.

Security

OSDI has several integrated security features. They are implemented using the z/OS RACROUTE interface, which provides program access to the z/OS System Authorization Facility, or SAF. The discussions and examples in this manual are based on IBM's z/OS Security Server (RACF) product, but any product that fully supports SAF can be used.

OSDI provides security processing during the following actions:

- An OSDI command is issued
- An address space binds to an OSDI-managed service
- A user requests a connection to an Oracle RDBMS instance with SYSOPER or SYSDBA privileges

- Certain types of users log on to an Oracle RDBMS instance for normal application processing

Command and bind authorization checking are part of the subsystem. When an OSDI command is processed, the subsystem performs a check to see if the user or console that is issuing the command is authorized to do so. The check is based on a resource name consisting of the subsystem name and the command verb. Similarly, subsystem bind processing checks to make sure that the user, address space, or task that is issuing the bind is authorized to access the target OSDI service that is specified in the bind request. The bind authorization check distinguishes between the two different types of bind discussed in the previous section. This is because the managed connection type of bind carries certain privileges that should not be made available to normal applications.

The SYSOPER/SYSDBA authorization check is performed by the database instance to which the connection is being made and is performed in addition to any bind authorization check made by the subsystem. This check uses a resource name consisting of the OSDI subsystem name, the database service name, and a fixed suffix string (either OPER or DBA).

In the first three of these authorization tests, if the SAF response is an indication that the associated resource is not defined, then the request is allowed. This means that RACF (or comparable) resource definitions are mandatory if command, bind, and Oracle SYSOPER/SYSDBA access are to be subject to SAF-based authorization controls.

In the fourth of these authorization tests, Oracle provides optional SAF-based password authentication at Oracle logon time for certain types of users.

Database Service

Starting an Oracle database service with an OSDI START command creates a z/OS address space and executes the JCL procedure that is specified in the service definition. In z/OS terms, the service runs as a system address space (similar to a started task or STC). The JCL procedure contains a single step that executes the OSDI database region control program. A few JCL DD statements are required. The region control program reads an input parameter file that specifies the name of the Oracle RDBMS kernel load module and several OSDI-specific operating parameters. The region control program then loads the Oracle kernel code into the address space and initializes various internal facilities.

If the database service has been defined to use multiple address spaces, and if the operating parameters indicate that additional address spaces are to be started immediately, then those address spaces are started automatically when initialization of the first address space is complete. Each auxiliary address space appears as a new STC executing the same JCL procedure as the first address space. Depending on how the service was defined, the new address spaces may have the same jobname as the first, or each may have a distinct jobname containing a three-digit suffix number. In addition to being started automatically, auxiliary address spaces can be started manually by issuing repeated OSDI START commands for the service. Each START command (after the first one) starts one more auxiliary address space, up to a maximum number that was specified when the service was defined.

When a database service uses multiple address spaces, client sessions are distributed more or less uniformly across those spaces. Service address space selection is performed by the subsystem during the processing of a client bind request. This is completely transparent to client software. No mechanism exists for a client bind request to designate or control the server address space to which it is assigned. Once a client session is bound to a given address space, the client remains there until it either disconnects from the database or terminates. If multiple database connections are made from a single client address space (in, for example, a multi-tasking application or in one of the Oracle Access Manager products), then those connections are distributed among the service address spaces just as though they had come from different client address spaces.

The auxiliary address spaces of a multi-address-space service do not respond to operator commands. All command interaction with the service is through the first address space, referred to as AS1. After auxiliary address spaces are started, they cannot be stopped individually. The auxiliary address spaces terminate when the entire service is terminated. Operator interaction with a running database service uses the OSDI STOP command and the z/OS STOP (P) and MODIFY (F) system commands, which are processed only by AS1.

OSDI initialization of a database service does not include the generic Oracle database startup. With OSDI, that step (which is required to make the database instance usable by applications) is performed separately. After the service address space, or spaces, are fully initialized, Oracle SQL*Plus (or a comparable mechanism) must issue an Oracle startup request to complete the instance startup processing. Only one startup request is required regardless of the number of address spaces the service is using.

Database service address spaces can persist over multiple Oracle startup and/or shutdown requests. In other words, OSDI does not terminate the database address spaces when an Oracle database shutdown is performed.

Network Service

Like the database, an Oracle Net network service is started with an OSDI START command and executes as a system address space. It uses a different JCL procedure from the database service, executes a different program, and has different JCL DD statement requirements. The network service runs in a single address space only and does not require a separate startup step as is required by the database service. When the network service address space starts, it initializes network protocol tasks and other internal facilities and initiates listen operations for inbound clients on designated network endpoints.

The network service must be running to support both inbound (remote) client connections to z/OS servers. (Outbound clients on z/OS--ones who are connecting to remote Oracle instances--interact directly with the TCP/IP protocol rather than using the network service.) Some important changes exist in the way that inbound connections to z/OS servers are processed.

OSDI simplifies the networking implementation on z/OS and makes it behave in a manner more similar to Oracle Net on other platforms. Regardless of which z/OS database servers are to be accessed, the network service listens for inbound clients on a single endpoint for each protocol. Clients that are connecting via a given protocol specify the target z/OS server using a SID parameter that is part of the Oracle network address string. The network service does not need to know in advance which servers will be accessed by inbound clients because the network service locates servers using the SID. Nothing needs to be done to a database server to prepare it for access by network clients.

2

Configuring and Initializing the Subsystem

This chapter describes how you create a z/OS subsystem for OSDI. The following topics are included:

Overview

To run any Oracle products under OSDI, you must first create a z/OS subsystem. The subsystem provides the internal interfaces that OSDI uses to process operator commands, to manage the execution of database and network services, and to manage connections between address spaces. One OSDI subsystem can support any number of distinct database instances and network services.

Chapter 17, "Migration and Upgrade Considerations", covers issues related to compatibility of OSDI with previous releases of Oracle Database for z/OS.

Choosing a Subsystem Name and Command Prefix

The subsystem name is a one-character to five-character identifier. z/OS requires that subsystem names begin with an alphabetic or national character, while subsequent characters can be alphanumeric or national. The name must be unique.

The subsystem requires a unique character string prefix to distinguish commands issued to it from system consoles and other system command sources. By default, OSDI will use the subsystem name as the command prefix. You can override this default if you wish, and specify a special character or a different alphanumeric string. Any character string prefix that you choose must not duplicate, or be a leading subset of, the command prefix of any other subsystem. Also, the character string prefix must not match any native z/OS system command name or abbreviation. Oracle Corporation recommends using the subsystem name for the command prefix.

The Subsystem Parameter File

During initialization, the subsystem opens and reads a sequential data set to obtain bootstrap initialization parameters and, optionally, any OSDI commands that are to be issued immediately after the subsystem is initialized. The file is usually a member of a PDS, and is created, viewed, and edited with ISPF or a similar tool. It can have either fixed-length or variable-length records. If fixed-length records are used, then sequence numbers in the rightmost record positions (for example, columns 73-80 of 80-byte records) are ignored.

Oracle Corporation recommends creating a PDS specifically for OSDI parameters and using it for subsystem parameters as well as for parameter files that are used by other components. Ten tracks of primary disk space, three tracks of secondary space, and eight directory blocks should be sufficient for most installations. This data set must be accessible for the subsystem to initialize, so it should not be subject to HSM migration or similar involuntary moves.

Only a single record, called the INIT record, is required in the subsystem initialization file. The INIT record has the following format:

```
INIT (ORASSI[,cmd-prefix][,cmd-class][,bind-class])
```

- INIT

  The word INIT can begin in position 1 or can be preceded by one or more blanks. It must be followed by one or more blanks and then by one to four positional parameters separated by commas and enclosed in a single pair of parentheses. The parentheses are required even if only the first positional parameter is coded. Do not include blanks in or among the positional parameters. If an optional positional parameter is not used, but a following one is, then a comma must be included for the unused parameter.

- First positional parameter

The first positional parameter is the name of the OSDI subsystem load module that was copied to a system linklist library during installation. This will normally be ORASSI, exactly as shown. This parameter is required.

- Second positional parameter (`cmd-prefix`)

  The second positional parameter is the command prefix for the subsystem. Any characters that are legal in a command prefix (except the comma and the left or right parenthesis) can be included. Do not enclose the prefix in apostrophes or quotes unless those are part of the prefix. The command prefix can be up to eight characters in length. If you omit this parameter, then the command prefix is assumed to be the subsystem name.

- Third positional parameter (`cmd-class`)

  The third positional parameter is the System Authorization Facility (SAF) resource class name that is to be used when authorizing access to OSDI commands. This parameter should be specified if you have created a specific resource class for command authorization as discussed in *Oracle Database Installation Guide for IBM z/OS*. If you omit this parameter, then OSDI uses the FACILITY class for command authorization requests.The following INIT record provides an example in which the SAF resource class names have been specified while the command prefix is allowed to default:

  ```
  INIT (ORASSI,,$ORACMD,$ORACONN)
  ```

- Fourth positional parameter (`bind-class`)

  The fourth positional parameter is the SAF resource class name that is to be used when authorizing client address spaces during bind to a service and when authorizing a local database connection with SYSOPER or SYSDBA privileges. This parameter should be specified if you have created a specific resource class for bind and connect authorization as discussed in the *Oracle Database Installation Guide for IBM z/OS*. If you omit this parameter, then OSDI uses the FACILITY class for bind and connect authorization requests.

The following INIT record provides an example in which the SAF resource class names have been specified while the command prefix is allowed to default:

```
INIT (ORASSI,,$ORACMD,$ORACONN)
```

OSDI Commands in the Subsystem Parameter File

The INIT record is the only item that is required to be in the subsystem initialization file. As a convenience, however, you can include OSDI commands in the initialization file immediately after the INIT record. Usually, you will have a number of OSDI DEFINE commands there. You might also have OSDI START commands for any services that you always want started as soon as possible during an IPL. Oracle Corporation recommends that you place into the parameter file both the DEFINE SERVICEGROUP command and the DEFINE SERVICE commands for commonly-used services.

The purpose and complete syntax of all OSDI commands is covered in Appendix A, "OSDI Subsystem Command Reference". You should consider the following items regarding commands that are supplied in the subsystem parameter file:

- Do not include the subsystem command prefix. All commands in the subsystem parameter file apply to the subsystem which is being initialized and is reading the file.
- Each command must begin on a new record. The command verb can begin in position 1 or it can be preceded by blanks.
- A command can be continued by including a hyphen (or minus sign) as the last non-blank character on a record (excluding any sequence number). The continuation can begin in position 1 of the next record or it can be preceded by blanks. You cannot continue (split across records) a command parameter that is enclosed in apostrophes. The continuation hyphen is not interpreted as part of the command.

As commands in the parameter file are processed by the subsystem, the command images are not displayed on the console log, but the subsystem response messages appear as if the commands were entered at a console.

Initializing the Subsystem

Before attempting to initialize the subsystem, the required OSDI subsystem modules must reside in the system linklist library. For more information, refer to the *Oracle Database Installation Guide for IBM z/OS*.

The subsystem can be initialized in either of two ways:

1. At system IPL, based on an entry that you add to the IEFSSN*xx* member of SYS1.PARMLIB
2. At any other time by using the SETSSI ADD system operator command

Oracle Corporation recommends that you add regularly-used subsystems to the IEFSSN*xx* member so that they are initialized correctly and automatically at every IPL. Use the SETSSI ADD system operator command when necessary, such as when you first install OSDI and want to try bringing up a subsystem without having to IPL.

Because OSDI uses z/OS dynamic subsystem interfaces, the IEFSSNxx entry for an OSDI subsystem must use the newer keyword parameter format, not the old positional format.

Assuming that you have chosen ORSS as your OSDI subsystem name and that the subsystem parameter file is member SSP01 of the data set ORACLE.ORA1.PARMLIB, an appropriate IEFSSN*xx* entry would be similar to the following:

```
SUBSYS SUBNAME(ORSS) INITRTN(ORASSINI)
  INITPARM('ORACLE.ORA1.PARMLIB(ORSS)')
```

In the above example, ORASSINI is the name of the subsystem's initialization routine. This module was copied to a system linklist library during OSDI installation. It must be specified exactly as shown.

The subsystem parameter string (INITPARM keyword) must specify the data set name and, if applicable, the member name of the subsystem parameter file containing the INIT record and optional commands. If no member name is supplied, then this must be a sequential (DSORG=PS) data set.

After updating IEFSSN*xx*, an IPL is necessary to process the added entry. If you do not want to wait for an IPL, or if other circumstances exist in which a subsystem must be created without an IPL, then you can use a SETSSI ADD system operator command equivalent to the following:

```
SETSSI ADD,S=ORSS,I=ORASSINI,P='ORACLE.ORA1.PARMLIB(ORSS)'
```

This example uses the minimal keyword abbreviations. The longer forms, SUBNAME (for S), INITRTN (for I), and INITPARM (for P), can be used if desired.

Caution: Oracle Corporation recommends that you use caution when entering the SETSSI ADD command. If you make a mistake in the parameter string (such as misspelling the data set or member name), then the subsystem will fail to initialize, and the subsystem name that you specified will be unusable until the next IPL.

Examples

The following code represents the contents of a subsystem initialization file that includes OSDI commands to define the service group and several services. The file ends with an OSDI SHOW command that will display the definitions in the system log.

```
INIT (ORASSI,ORSS)
DEFINE SERVICEGROUP SSID(ORSS) DESC('Oracle OSDI Subsystem')
DEFINE SERVICE ORAPROD  DESC('Oracle Production DB') -
  TYPE(ORA) PROC(ORADB01) JOBNAME(OPROD*)  MAXAS(8) -
  SID(ORA1) PARM('ORACLE.ORA1.PARMLIB(ORAPROD1)')
DEFINE SERVICE ORATEST DESC('Oracle Test DB') TYPE(ORA) -
  PROC(ORADB01) SID(ORAT) -
  PARM('ORACLE.ORA1.PARMLIB(ORATEST)')
DEFINE SERVICE ORANET DESC('Oracle Net') -
  TYPE(NET) PROC(ORANET01)  -
  PARM('HPNS PORT(1521)')
SHOW SERVICEGROUP LONG
```

Consider this file to be member SSP01 of ORACLE.V10G.PARMLIB as in the earlier examples, and assume that the subsystem will be initialized with a SETSSI ADD command using subsystem name ORSS. The z/OS system log that results from the SETSSI ADD command would be similar to the following:

```
SETSSI ADD,S=ORSS,I=ORASSINI,P='ORACLE.ORA1.PARMLIB(ORSS)'
MIS0020I Initialization of Oracle subsystem ORSS complete
MIS0196I Service group ORSS defined
MIS0198I Service ORAS10    defined
MIS0198I Service ORAN10    defined
MIS0195I Service group ORSS (OSDI Oracle 10G Subsystem - ORSS)
         Mode=*SYS   , Systems=*ALL
         Service ORAN10    Type NET  (Oracle V10G Net Service)
         Service ORAS10    Type ORA  (Oracle V10G RDBMS Service)
MIS0193I Service ORAN10    starting
MIS0193I Service ORAS10    starting
```

At this point, a subsystem has been initialized, and three services have been defined. Configuring and operating a database service and network service are described in Chapter 3, "Configuring a Database Service and Creating a New Database" and Chapter 8, "Oracle Net".

After Initializing the Subsystem

When you have successfully initialized the subsystem, you can proceed to configure and start Oracle products that will run under its control. Before doing so, you may want to enable security mechanisms that the subsystem provides. These security mechanisms control access to subsystem commands, and they control program access to the services that you define in the subsystem. Refer to Chapter 9, "Security Considerations", for information about these and other security features.

3 Configuring a Database Service and Creating a New Database

After you have created an OSDI subsystem, you can configure and initialize one or more Oracle databases to run under that subsystem. This chapter describes how to set up OSDI definitions, JCL procedures, parameter files, and other z/OS-specific items required by an Oracle database instance. The three chapters, Chapter 4, "Defining z/OS Data Sets for the Oracle Database", Chapter 5, "Operating a Database Service", and Chapter 6, "Database Backup and Recovery", provide additional details on Oracle database files, database service operation, and database backup and recovery. We suggest that you read these chapters before configuring a new database service, and that you review Chapter 9, "Security Considerations", for information about z/OS security features that affect an Oracle database service.

If you are migrating an existing Oracle database on OS/390 or z/OS to Oracle Database 10*g* for z/OS, you will not be creating a new database as described in this chapter. Refer to Chapter 17, "Migration and Upgrade Considerations", for details on migrating your existing database. If you are new to OSDI, read this chapter to learn how OSDI differs from the MPM subsystem as far as database configuration is concerned.

The following topics are included:

- Overview
- Database Service Definition
- Database Region JCL
- Database Region Parameters
- Oracle Initialization Parameter Considerations
- Pre-Allocating Database Files

- Configuring z/OS Security
- Configuring for Shared Servers
- Creating the Database
- Creating an Oracle Database Instance

Overview

To create an Oracle database instance under OSDI, you must first define the instance as a service using the OSDI `DEFINE SERVICE` command. In addition to defining the service, some other items must be set up before the service can be started: a JCL procedure, several parameter files, and possibly security resource definitions.

After you have defined the instance as a service and set up the additional items, you can start the service, which creates one or more z/OS address spaces based on controls that you have specified. After the address spaces are initialized, you must run Oracle SQL*Plus (or a similar tool) to perform the Oracle database startup function. When the startup is complete, you can use the same tool to issue the `CREATE DATABASE` SQL statement. This statement causes the Oracle server to create the VSAM linear data sets that comprise a database (if you chose not to pre-allocate them) and to initialize their contents. After the database is created, a series of SQL scripts is executed to create the Oracle server's internal database objects (tables, views, stored PL/SQL procedures, and so forth). After the execution of the scripts is complete, your database is ready to use.

A description of the configuration process is included in this chapter on page 3-29.

Database Service Definition

The OSDI `DEFINE SERVICE` command is described completely in Appendix A, "OSDI Subsystem Command Reference". Here, we cover `DEFINE` parameter considerations that are specific to an Oracle database service.

Service Name

The service name for a database can be anything that you want within the content limitations described in Appendix A. By default, OSDI will use the service name as the SID for the service. (The SID is an identifier that end users or application developers must supply to connect an application to a particular database.) The SID can be specified separately, however, and is not required to be the same as the service name.

Although OSDI permits service names up to eight characters long, the name you use for a database service should be seven characters or less due to a length limitation on what is stored in the database control file. The OSDI service name appears in the Oracle data dictionary view `V$INSTANCE` in column `INSTANCE_NAME`.

Note: If you specify a service name that is the same as any existing subsystem name in your system (Oracle database or otherwise), then you must also specify a `JOBNAME` parameter that is not the same as any existing subsystem. If you do not use unique names, then OSDI starts the service using the service name as the job identifier. When z/OS processes a start for an address space whose job name or job identifier matches a known subsystem, the job runs under control of the master subsystem instead of under control of JES.

Caution: Running OSDI services under the master subsystem is not supported. This situation must be avoided by making sure that the service runs with a job name or a job identifier that is not the same as any subsystem name.

TYPE

The `TYPE` parameter for a database service must be specified as ORA.

PROC

This parameter specifies the name of a service JCL procedure that you will place in one of your system procedure libraries. The procedure need not exist when `DEFINE SERVICE` is issued, but it must be in place before the service is started. The procedure name can be anything that you choose or that the naming standards of your installation require. The requirements for this procedure are discussed in section "Database Region JCL" on page 3-5.

PARM

The `PARM` for a database service specifies the name of a z/OS data set containing service initialization parameters. These are z/OS-specific parameters (not the Oracle RDBMS `init.ora` file startup parameters) and are described in the section "Database Region Parameters" on page 3-9. Typically, `PARM` will specify a member of a PDS (Partitioned Data Set) that is used for various Oracle parameter files. If no member name is included in the PARM string, then the specified data set must be sequential (`DSORG=PS`).

MAXAS

If you want to exploit the multiple-address-space server features of OSDI, then you should specify the `MAXAS` parameter on `DEFINE SERVICE` with a value greater than the default of 1. This sets the maximum number of address spaces for the service, which may be greater than the number started when the service is first brought up. (The number of address spaces to start initially is a database region parameter.) This parameter can be altered with OSDI commands as long as the database service is not active.

JOBNAME

When you run a database service with multiple address spaces, the `JOBNAME` parameter of `DEFINE SERVICE` can be used to cause each address space to have a distinct jobname. Although this is not required, it may be desirable if you use z/OS facilities (such as RMF) that distinguish address spaces by jobname. To do this, specify `JOBNAME(`*`name`*`)`, where *`name`* is a one-character to five-character jobname prefix followed by an asterisk, as shown. As each address space is started, OSDI substitutes a three-digit address space counter for the asterisk (001, 002, and so on) to produce the final jobname. You can also use `JOBNAME` to cause the service to run with a jobname different from the service name (which is used by default).

As discussed in the Note in the preceding page, you must specify a `JOBNAME` parameter if the service name matches any existing subsystem name in your z/OS system.

SID

The SID parameter specifies a unique identifier for the service. It is a critical element in the process that is used by Oracle database applications to specify the instance to which they need to connect. (Inbound network clients specify a SID in the Oracle database network address string that must match the SID that is specified in `DEFINE SERVICE`. Local z/OS clients connecting via cross-memory specify the SID in any of several ways.) Although the SID can be up to eight characters long, you may want to specify a SID that is four characters or less in order to enable a z/OS-specific feature that local z/OS clients can use to specify a target database. This feature relies on a dummy JCL DD statement (or TSO allocation) whose DD name begins with "`ORA@`" and ends with a one-character to four-character SID of the target database instance. If you choose a SID longer than four characters (or allow it to default to a service name that is longer than four characters), this feature is not usable.

Although you can issue the OSDI `DEFINE SERVICE` command via a z/OS system console or similar facility, you should put definition commands for services that you use regularly into the OSDI subsystem parameter file, after the `DEFINE SERVICEGROUP` command. This ensures that the service is always defined correctly and automatically when the subsystem is initialized (normally at system IPL). In the following sample database `DEFINE SERVICE` command, the command prefix has been omitted and continuation hyphens have been included as though the command were in the subsystem parameter file:

```
INIT (ORASSI,ORSS)
DEF SVG SSID(ORSS) DESC('OSDI Oracle 10G Subsystem - ORSS')

DEF SRV ORAS10 PROC(ORA1S10) TYPE(ORA) MAXAS(1) -
 DESC('Oracle V10G RDBMS Service') -
 SID(ORA1) PARM('ORACLE.ORA1.PARMLIB(ORSSPARM)')

DEF SRV ORAN10 PROC(ORA1N10) TYPE(NET) -
 DESC('Oracle V10G Net Service') -
 SID(ORAN) PARM('HPNS PORT(1501) ENCLAVE(SESS)')

SHOW SERVICEGROUP LONG

START  ORAS10
START  ORAN10
```

Database Region JCL

Defining a database service requires you to specify a JCL procedure name in a system procedure library. You must create the procedure before you try to start the service, and the procedure must invoke the OSDI database region program with an EXEC statement such as the following:

```
// EXEC PGM=ORARASC,REGION=0M
```

`REGION=0M` is specified to ensure that the server can allocate as much private virtual memory as it needs. Some z/OS systems may prohibit or alter a REGION parameter such as this, so you might want to check with your systems programmer to make sure that the system will accept your `REGION` parameter.

A z/OS exit called IEFUSI might be installed on your system. The IEFUSI exit prevents started tasks or batch jobs from getting the maximum region size when `REGION=0M` is specified. If an IEFUSI exit is implemented, it is specified in the

SMFPRM*xx* member of SYS1.PARMLIB that is used during z/OS initialization. To effectively run the Oracle database with an IEFUSI exit installed, ensure that the exit is coded to allow batch jobs or started tasks with the names of your Oracle regions to allocate a large amount of virtual memory above the 16 MB line.

Because Oracle allocates only the amount of memory it needs, you can safely allow Oracle to allocate any amount of memory up to the two gigabyte limit per address space that is imposed by 31-bit addressing conventions.

Note that no other EXEC statement parameters are needed. The PARM parameter of EXEC is not used by the database region program.

Changing a service's JCL procedure after starting one or more address spaces for the service, and then starting another address space (to use the changed JCL), is not supported.

In addition to the EXEC statement, the procedure will need several DD statements, as follows:

ORA$ENV: This DD statement is optional. When used, it specifies a sequential file or PDS member containing environment variable assignment statements. Environment variables are used to supply operating parameters to certain Oracle database product features. Reliance on environment variables and considerations for setting them are discussed in feature-specific chapters of this manual as well as in the *Oracle Database User's Guide for IBM z/OS (OS/390)*. The data specified by ORA$ENV is read only at database service startup. Therefore, in order for changes to the data set to take effect, the service must be stopped and started.

Be aware that the global environment variable file is not read by the Oracle database server. All environment variable settings for the server must be supplied through ORA$ENV. For more information on the global environment variable file, refer to Chapter 7, "Oracle Database Administration Utilities".

> **Note:** The ORACLE_HOME environment variable (referring to the ORACLE_HOME directory name under HFS, specified during installation) is required for components that run in a UNIX System Services shell environment, such as Oracle JVM, Oracle Text, and the time zone feature.

ORA$FPS: This DD statement specifies a sequential file or PDS member containing z/OS-specific parameters that control data set processing in the Oracle server. These parameters are organized by type of file (such as tablespace, control, online log, and so forth), and they primarily pertain to creation processing when the Oracle server invokes the `IDCAMS` utility or dynamic allocation to create a z/OS data set. Considerations and syntax rules for the `ORA$FPS` parameter file are covered in "Server File Management Parameters". The `ORA$FPS` DD is optional. If you omit it, then server file creation operations may fail unless your installation has DF/SMS ACS routines that supply defaults for data set creation parameters. At database service startup, data specified by `ORA$FPS` is read and checked. Any errors are reported and ignored. Valid entries are loaded as server file management parameters. After database service startup, a new set of server file management parameters can be loaded from the updated `ORA$FPS` specification by using the `REFRESH FPS` command, as described in Chapter 5, "Operating a Database Service".

> **Note:** When this DD statement is omitted, an IEC130I message may appear in the system log during service address space initialization. This is normal.

ORA$LIB: This DD statement specifies a non-authorized load library from which non-executable (data) modules are fetched. The modules contain NLS data objects and messages that are associated with Oracle NLS internationalization features. Normally these modules are installed in the OSDI MESG data set, for example `ORACLE.V10G.MESG`. The `ORA$LIB` DD statement is optional: if you omit it, then the Oracle server attempts to fetch messages and NLS data objects modules from `STEPLIB`. Do not concatenate a non-APF-authorized MESG data set to `STEPLIB` in lieu of specifying `ORA$LIB`.

> **Note:** When this DD statement is omitted, an IEC130I message may appear in the system log during service address space initialization. This is normal.

ORAPASSW: This DD statement is optional. It specifies a VSAM linear data set that has been initialized with the `ORAPWD` utility. This file contains encrypted passwords and is used only to authenticate a client who is connecting as `SYSDBA` or `SYSOPER`. The use of this file is described in "Security Considerations" in the section "Controlling Access to Database SYSDBA and SYSOPER Privileges" and the `ORAPWD` utility is discussed in Chapter 7, "Oracle Database Administration Utilities".

SNAPCF: This DD statement is optional. When used, it specifies a VSAM linear data set that contains a copy of the database control file. The considerations for this file are discussed in Chapter 6, "Database Backup and Recovery".

SQLBSQ: This DD statement specifies an input file containing the Oracle database "bootstrap" SQL script. It is read only during an Oracle database cold start (`CREATE DATABASE` SQL statement) and is therefore required only when a cold start is planned. When specified, it usually designates the SQLBSQ member of a partitioned data set dedicated to SQL scripts. This data set was created during Oracle product installation.

SQLNET: This DD statement specifies an input file containing Oracle Net parameters. It is required if the Oracle instance uses any of the following:

- Network data encryption
- Network activity tracing
- Altering of default Oracle Net file names
- Outbound database links whose Oracle Net addressing requires access to an Oracle database Names server

Refer to Chapter 8, "Oracle Net", for additional information.

STEPLIB: This DD statement must specify the APF-authorized Oracle `AUTHLOAD` library that was populated during installation. The IBM LE/370 runtime library must be concatenated to it unless your installation has put LE/370 runtime into the system linklist. A typical name for the LE/370 runtime library is `SYS1.SCEERUN`, but it may have a different name in your system.

SYSPRINT: This DD statement is optional. When used, the Oracle database instance alert log is written to it. The alert log is a sequential text file containing status messages that are related to the operation of the database instance, including startup and shutdown information, log file switches, archive operations, and certain types of error condition. The alert log is also used to log some z/OS-specific events, including IDCAMS utility output associated with database file creation and deletion. Regardless of the number of server address spaces, an Oracle database instance has only one alert log, which is opened by the first server address space (AS1). Alerts that are generated by sessions in other address spaces are routed to AS1.

You can specify a sequential (`DSORG=PS`) disk data set or a spool file (`SYSOUT`) for this DD. If you omit the `SYSPRINT` DD, the alert log is dynamically allocated as a disk data set or spool file according to the ALERT_DSNAME region parameter, discussed in "Database Region Parameters" in Chapter 3, "Configuring a Database Service and Creating a New Database".

If you specify a disk data set for SYSPRINT and an error occurs while it is being written (including an out of space condition), an alert log switchoccurs. Refer to Chapter 5, "Operating a Database Service" Managing the Alert Log for additional information on Oracle alert log switching.

TNSNAMES: This DD statement specifies an input sequential file or PDS member containing Oracle Net name/address assignments. It is required if the Oracle instance uses database links (connections to other Oracle database instances) whose USING clause specifies an Oracle Net service name rather than an explicit Oracle Net address. If you are using external routines or shared servers, refer to "Step 2: Edit the Server Sqlnet.ora File" in Chapter 8, "Oracle Net" to add the correct entries.

Sample Database Region JCL Procedure The following is an example of a JCL procedure for a database region:

```
//ORA1S10 PROC
//*-----------------------------------------------------------------*
//*--   ORACLE DATABASE SERVICE PROCEDURE                         --*
//*--                                                             --*
//*-----------------------------------------------------------------*
//IEFPROC  EXEC PGM=ORARASC,REGION=0M
//STEPLIB  DD  DISP=SHR,DSN=ORACLE.V10G.AUTHLOAD
//ORA$LIB  DD  DISP=SHR,DSN=ORACLE.V10G.MESG
//*
//ORA$FPS  DD DISP=SHR,DSN=ORACLE.ORA1.PARMLIB(ORA1FPS)
//ORA$ENV  DD DISP=SHR,DSN=ORACLE.ORA1.PARMLIB(ORA1ENV)
//TNSNAMES DD DISP=SHR,DSN=ORACLE.ORA1.PARMLIB(TNSNAMES)
//TNS@ORAN DD DUMMY
```

Database Region Parameters

OSDI database region parameters are supplied in a data set whose name is specified as the PARM string in the service definition. This will typically be a member of a PDS. Because the data set name is supplied via the service PARM mechanism, no DD statement is coded in the region JCL. The data set is dynamically allocated, opened, and read when the service is started. Changing parameters in the data set has no effect until the service is stopped and restarted.

Region parameters are read independently by each address space of a multi-address space server. Adding, removing, or changing parameters between the starting of one address space and later starting of another is not supported.

The OSDI database region parameters consist of a parameter name followed by the parameter value in parentheses. Each parameter has a long descriptive name and a shorter name of eight characters or less. Each record may contain only one parameter. No continuation is allowed. Records beginning with an asterisk (*) are treated as comments and are ignored. Embedded spaces and all characters after the closing parenthesis are ignored.

ALERT_DSNAME | ADSN

`ALERT_DSNAME` specifies a filespec for the Oracle database alert log for the purpose of alert log switching. The format for this parameter is as follows:

```
ALERT_DSNAME ( filespec )
```

The *`filespec`* value can be a `SYSOUT` type or a data set name type with embedded system symbols that will guarantee a unique data set name on each use. Using date and time system symbols is recommended in a data set name filespec.

Examples:

```
  ALERT_DSNAME(ORACLE.&ORASRVN..ALERT.D&LDATE..T&LTIME)
  ADSN(//S:Z,,DBOPS01)
```

If you omit this parameter, alert log switches use a default `SYSOUT` specification `//SYSOUT:*`. For more information on alert log switching, refer to "Managing the Alert Log" in Chapter 5, "Operating a Database Service".

ALERT_MAX | AMAX

`ALERT_MAX` specifies:

```
ALERT_MAX ( size )
```

The *`size`* value is the number of data bytes (sum of logical record lengths) written to the alert log. This value can be specified as a number, *`n`*, *`n`*`K` (denoting a multiplier of 1024), or *`n`*`M` (denoting a multiplier of 1,048,576). The writing of an alert log record that would exceed this size causes an automatic alert log switch before the new record is written. This happens without regard for the sequence or interrelationship of alert log messages; for example, the switch can occur between a pair of related messages.

The default value is `0`. If you specify `0`, no automatic switching is done. Specifying a value less than 65536 (64 KB) is not recommended for this parameter.

ALERT_MIN | AMIN

`ALERT_MIN` specifies:

```
ALERT_MIN ( size )
```

The *`size`* value is the number of data bytes (sum of logical record lengths) written to the alert log. This value can be specified as *`n`*, *`n`*`K` (denoting a multiplier of 1024), or *`n`*`M` (denoting a multiplier of 1,048,576). If an alert log switch is requested (for example, using a `LOGSWITCH` service command without the `FORCE` option), the request is honored only if the size of the current alert log exceeds this value.

The default value is `0`. If you specify `0`, no minimum size checking is done, and all alert log switch requests are carried out.

CLUSTER_ENABLE | CLUSTER

The `CLUSTER_ENABLE` parameter specifies whether to enable or disable the Oracle Real Application Clusters option in the Oracle kernel. The format is as follows:

```
CLUSTER_ENABLE ( YES | NO )
```

The default value is `NO`. Setting `CLUSTER_ENABLE` to `YES` causes the `ORARASC` module to activate a subtask that handles the IBM/XCF group management calls required by Oracle Real Application Clusters.

DSN_PREFIX_DB | ORAPREFD

The `DSN_PREFIX_DB` parameter supplies a constant string that is associated with the `&ORAPREFD` system symbol. The `&ORAPREFD` system symbol can be used to form the high-level (leftmost) qualifier of z/OS data set names generated by the Oracle server. The format is as follows:

```
DSN_PREFIX_DB ( dsn_prefix )
```

The `dsn_prefix` value is a valid one-character to eight-character data set name qualifier that conforms to your installation's requirements. In most cases, this will be the qualifier that is used for all Oracle database files associated with this instance. For example:

```
DSN_PREFIX_DB(ORADB01)
```

`DSN_PREFIX_DB` has no default value. If you omit this parameter, certain situations in which the Oracle server generates "default" filenames will produce errors. Refer to Chapter 4, "Defining z/OS Data Sets for the Oracle Database", for more information.

IDLE_TIMEOUT | ITIMEOUT

The `IDLE_TIMEOU` parameter parameter sets a timeout value for idle sessions. Sessions that are idle for a period longer than the interval set are terminated and all resources released. The format is as follows:

```
IDLE_TIMEOUT ( time_interval )
```

The *`time_interval`* value is the timeout value specified as *`nnn`* or *`nnn`*S for seconds or *`nnn`*M for minutes. The default is no timeout value. The maximum value is 604,800 seconds or one week. The timeout value set is a minimum approximation and a session may be idle for some additional seconds or minutes before it is terminated. When a session is using a dedicated TCB, as is the case with the Transparent Gateway products, the task is terminated with an S222 completion code. Clients with connections to timed-out sessions may see a variety of errors if they attempt to continue.

INIT_ADR_SPACES | INTADSPC

`INIT_ADR_SPACES` controls how many auxiliary address spaces are started. The format is as follows:

```
INIT_ADR_SPACES ( number_of_address_spaces )
```

The *`number_of_address_spaces`* value is the number of address spaces to start. The default is 1, which starts only the control address space (AS1). The maximum is the number that was specified for `MAXAS` on the associated `DEFINE SERVICE` command for the database service.

INIT_STACK_SIZE | INTSTKSZ

`INIT_STACK_SIZE` controls the size of the C stack that is allocated for each session. The format is as follows:

```
INIT_STACK_SIZE ( init_size )
```

The *`init_size`* value determines the initial size of the C stack. This value can be specified as *n* or *n*K. The default is 128K. For more information on `INIT_STACK_SIZE`, refer to "User Stack Area in z/OS" in Chapter 15, "Oracle Database Performance".

If the RDBMS Java system will be initialized, and if Java stored procedures will be used, then the value of *`init_size`* should be at least 256K.

LOGON_AUTH | LGNAUTH

`LOGON_AUTH` specifies how the Oracle server interacts with a SAF-based external security product when processing Oracle logons for users defined as `IDENTIFIED EXTERNALLY`. The format is as follows:

```
LOGON_AUTH ( auth )
```

The *`auth`* value can be specified as follows:

Value	Description
`NONE`	IDENTIFIED EXTERNALLY not allowed (no interaction)
`SAF`	perform built-in `SAF RACROUTE` verification
`exitname`	call an installation-supplied logon exit; *`exitname`* is the one-character to eight-character load module name of the exit

If *`exitname`* is specified, then it must reside in the system linklist, or in an APF-authorized library that is part of the server region `STEPLIB` concatenation. The default is `NONE`.

Examples:

```
LOGON_AUTH(NONE)
LOGON_AUTH(RACFSMPO)
```

For more information about Oracle logon authorization, refer to Chapter 9, "Security Considerations".

MAX_SESSION_MEM | MAXSMEM

The `MAX_SESSION_MEM` parameter specifies a hard limit on the amount of virtual memory that a single database session can allocate. The format is as follows:

```
MAX_SESSION_MEM ( session_memory )
```

The *`session_memory`* value is the maximum amount of virtual memory that a single database session can allocate. This value can be specified as *n*, *n*K (denoting a multiplier of 1024), or *n*M (denoting a multiplier of 1,048,576). The default is zero (0), which means no session limit is imposed.

This parameter is useful for stopping a "runaway" session that is allocating excessive amounts of memory due, perhaps, to problems with application design. This pertains only to session-private C stack and "heap" memory allocated during Oracle server processing. It does not include SGA (System Global Area) memory used by a session nor internal memory allocations done by the implementation.

Care must be taken in choosing a limit, particularly where certain database administration operations might be affected. The "catalog build" step of new database creation requires as much as 96 MB of session memory and may fail if this parameter is set to a lower value. Omit this parameter or set it to a higher value during new database creation; you can change it to a lower value afterward if desired. In the current product release, a normal database startup requires up to 16 MB of session memory, so do not set this parameter to a value less than 16MB.

MAX_SESSIONS | MAXSESS

The `MAX_SESSIONS` parameter limits the number of sessions that can be scheduled in an address space. The format is as follows:

```
MAX_SESSIONS ( number_of_sessions )
```

The *`number_of_sessions`* value is the maximum number of sessions per address space. This value can be specified as *n* or *n*K. The default is 1024. The number of sessions that can be supported in an address space depends on the complexity of the work. Limiting the number of sessions per address space reduces the chances of session failure due to exhaustion of virtual storage. Refer to "Database Server Address Space Configuration" in Chapter 15, "Oracle Database Performance" for more information.

REGION_MEM_RESERVE | REGMRES

The `REGION_MEM_RESERVE` parameter specifies the amount of private area memory in the server address space to be "reserved" for implementation and z/OS use (not available for the SGA and Oracle session-private purposes). The format is as follows:

```
REGION_MEM_RESERVE ( region_memory )
```

The *region_memory* value is the amount of private area memory reserved. This value can be specified as *n*, *n*K (denoting a multiplier of 1024), or *n*M (denoting a multiplier of 1,048,576).

During initialization, each server address space calculates the total available private area memory and subtracts the reserve amount from it. The result is the aggregate limit for the SGA and for all session memory requests in that address space.

The default is zero (0), which means that no aggregate limit applies. In this case, it is possible for SGA and session memory requests to exhaust the available private area of the address space, leading to unpredictable failures.

Thus, the reserve amount must be sufficient to accommodate internal implementation memory requrements as well as memory required by z/OS services used by Oracle, particularly Local System Queue Area (LSQA) memory, which is used by all database I/O operations. Because it is difficult to predict this amount for any given workload, the best strategy is to specify a relatively large reserve amount, such as 50M or more. This has the effect of reducing slightly the number of sessions that can be accommodated in a server address space. However, additional address spaces can be started, if necessary.

SERVER_LOADMOD | SRVRLMOD

`SERVER_LOADMOD` specifies the name of the service load module. The format is as follows:

```
SERVER_LOADMOD ( loadmod )
```

The *`loadmod`* value is the name of the load module to load. For the Oracle RDBMS, this is usually `ORACLE`. This parameter is required.

SMF_STAT_RECNO | SMFSTRCN

`SMF_STAT_RECNO` specifies the SMF record number to use. The format is as follows:

```
SMF_STAT_RECNO ( record_number )
```

The *record_number* value is the number of the desired record of Oracle SMF statistics. The default is zero (0). Otherwise, the value must be specified between 128 and 255 for this parameter. Example:

```
SMF_STAT_RECNO(204)
```

The collection and writing of Oracle SMF statistics records is controlled by this single parameter in the OSDI service parameter file. A zero (0) for this parameter indicates that no SMF statistics record is to be written. The SMF record number that is chosen must not be the same as the number that is used by any other z/OS software.

If this parameter is not specified, or if zero is specified, then no SMF statistics collection or recording is done. This saves some CPU overhead and saves the overhead of the SMF write itself (which is mostly asynchronous work done by the

SMF address space, the in-line overhead is mainly just moving data into SMF buffers). For more information about SMF, refer to Chapter 10, "Oracle SMF Data".

TRACE_DSNAME | TDSN

TRACE_DSNAME specifies the destination for Oracle RDBMS trace files. This includes normal traces requested by setting the session SQL_TRACE option to TRUE, as well as diagnostic traces generated automatically in certain error situations. The format is as follows:

```
TRACE_DSNAME ( filespec )
```

The filespec value is either a SYSOUT specification (including class, form, and JES destination) or a data set name.

A SYSOUT specification is of the form:

```
//SYSOUT:class,form,dest
```

as described in "Server File Name Syntax" in Chapter 4. When this is used, trace files are dynamically allocated SYSOUT data sets. In a multi-address space service, the trace file for a given database session is allocated in the address space that hosts the session. Thus, SYSOUT trace files can appear in all server address spaces. For example, traces written to SYSOUT class X, form AA01, would be written as:

```
TRACE_DSNAME(//SYSOUT:X,AA01)
```

As an alternative to a SYSOUT specification, you can specify a data set name. Because each trace file created as a data set must have a unique data set name, the supplied value must include system symbols that guarantee uniqueness. Refer to Appendix C, "Oracle Database for z/OS System Symbols" for more information.

To guarantee uniqueness, use some combination of the session identifier (&ORASESST) system symbol, date (&LYYMMDD), and time (&LHHMMSS). Also use high-level qualifier(s) that are appropriate for your installation. This will avoid the possibility of duplicating trace data set names generated in other Oracle instances you run. All components of the string must resolve to produce a name that is valid for a z/OS sequential data set. For example:

```
TRACE_DSNAME(ORA3A.TRACE.D&LYYMMDD..T&LHHMMSS..&ORASESST)
```

The allocation parameters for Oracle trace data sets are obtained from the DBTR file group of the server file management parameters, discussed in Chapter 4, "Defining z/OS Data Sets for the Oracle Database".

If this parameter is omitted or fails to produce a valid, unique data set name, all Oracle trace files are written to the default `SYSOUT` class associated with the server region.

Oracle Initialization Parameter Considerations

Oracle server initialization parameters, or `init.ora` file parameters, can be supplied in several ways. The oldest mechanism, still supported for backward compatibility, is to use parameters in a text file that is read by the utility doing the startup (usually SQL*Plus, but possibly RMAN or others).

This occurs when you issue "`STARTUP PFILE=filespec...`". The filespec you supply on this command can be a sequential data set, a PDS member, an instream (DD *) data set, or a file in the POSIX HFS; the z/OS userid that is running the utility must be authorized to open it.

If you omit `PFILE=` from the `STARTUP` command, no file is opened in the utility session, and `STARTUP` processing in the Oracle database server address space attempts to open a parameter file. The server first tries to open an `SPFILE` (server parameter file), which is a database-type file, a VSAM linear data set (LDS), containing current parameter settings. You can create an `SPFILE` using the `CREATE SPFILE` SQL statement. One advantage to using an `SPFILE` is that the server can update it when you change parameters with `ALTER SYSTEM`, for example, saving you from having to remember to make such changes to a text PFILE. For more information on creating an `SPFILE` on z/OS, refer to the section "SPFILE" on page 3-21.

If the server cannot locate or open an `SPFILE`, it attempts to open a regular `PFILE` instead. The default filespec for this file on z/OS is `//DD:INITORA`, so in order to read a `PFILE` the server region JCL must contain an `INITORA` DD statement. A `PFILE` read by the server must be a sequential data set or a member of a partitioned data set. The z/OS userid associated with the server address space must be authorized to read this data set.

Considerations for most of the initialization parameters in this file are the same regardless of the operating system on which the associated Oracle database instance runs. However, some have z/OS-specific considerations, discussed in this section.

Use this section together with the other z/OS-specific documentation when choosing initialization parameters for an Oracle server on z/OS.

For a complete list of initialization parameters with z/OS- specific defaults or limits, refer to Table B–1, " Initialization Parameters with z/OS-Specific Defaults or Limits" inAppendix B, "Operating System Dependent Variables".

AUDIT_FILE_DEST

The AUDIT_FILE_DEST parameter specifies the desired SMF record type to which audit records will be written when the AUDIT_TRAIL initialization parameter is set to OS. For example, specifying AUDIT_FILE_DEST=205 causes Oracle Database for z/OS audit records to be written to SMF record type 205. The default for this parameter on z/OS is 0, which means no SMF records will be written.

When specifying a record type, you must select a user SMF record type that does not conflict with any other user record types. For more information, refer to Chapter 10, "Oracle SMF Data".

AUDIT_TRAIL

The AUDIT_TRAIL parameter enables or disables database auditing. Setting AUDIT_TRAIL=OS is required to inform Oracle Database for z/OS that operating system auditing is desired.

CONTROL_FILES

The CONTROL_FILES parameter specifies the name, or names, of one or more database control files that are specified using the file name syntax discussed in Chapter 4, "Defining z/OS Data Sets for the Oracle Database". When you are first creating a database, if you choose to let the Oracle server allocate your control files (instead of pre-allocating them yourself), this parameter will specify the VSAM linear data set names that do not yet exist but will be created during processing of the CREATE DATABASE command.

The following is a sample CONTROL_FILES parameter for z/OS using the full file name syntax:

```
CONTROL_FILES = "//'ORAPROD.ORADB1.CTL1'", "//'ORAPROD.ORADB1.CTL2'"
```

DB_BLOCK_SIZE

The DB_BLOCK_SIZE parameter is used as the default logical database blocksize for all tablespaces that do not request a different logical blocksize with the BLOCKSIZE option of CREATE TABLESPACE. DB_BLOCK_SIZE can be specified as 4096 (4 KB), 8192 (8 KB), 16384 (16 KB), or 32768 (32 KB).

The default for this parameter on z/OS is 4096. Regardless of what logical blocksize you use, Oracle database files on z/OS always have a physical blocksize of 4096 bytes. For more information on database logical blocksize, refer to the section "Pre-Allocating Database Files".

DB_CREATE_FILE_DEST; DB_CREATE_ONLINE_LOG_DEST_n

These parameters are associated with Oracle Managed Files (OMF). OMF simplifies database administration by making the Oracle server responsible for naming, creating, and deleting the VSAM linear data sets comprising a database. Refer to "Oracle Database Files" in Chapter 4, "Defining z/OS Data Sets for the Oracle Database" and the *Oracle Database Administrator's Guide* for more information on OMF.

On z/OS, these parameters supply character strings that are used as the left-hand portion of the data set names generated by the server when an OMF file is created. Valid z/OS and Oracle-specific system symbols can be included. The maximum length permitted after any system symbols are resolved is 23 characters for `DB_CREATE_FILE_DEST` and 29 characters for `DB_CREATE_ONLINE_LOG_DEST_`*n*. To be usable for z/OS data set name generation these strings must end with a period after any symbol substitution has been done. The following are some examples:

```
DB_CREATE_FILE_DEST = "ORACLE.ORADB01."
DB_CREATE_ONLINE_LOG_DEST_1 = "&ORAPREFD..&ORASRVN..M1."
DB_CREATE_ONLINE_LOG_DEST_2 = "&ORAPREFD..&ORASRVN..M2."
```

DB_FILE_NAME_CONVERT; LOG_FILE_NAME_CONVERT

These parameters are used in conjunction with the standby database availability feature and in certain point-in-time recovery situations. They cause the server to convert database and log file names that are read from the control file by replacing a portion of the original file name with another value.

To use the standby feature and the `DB_FILE_NAME_CONVERT` parameters, your database data file data set names must share a common naming convention. The easiest way to meet this requirement is to use the same high-level qualifier (or qualifiers) for all database data file names. The same logic applies to the `LOG_FILE_NAME_CONVERT` for the redo log files.

For example, if all of your primary database data files have data set names beginning with "`ORA5.DBPRIM.`" and you choose "`ORA5.DBSTNDBY.`" as the prefix for database files in your standby database, then to effect the name conversion for the standby database, you would specify:

```
DB_FILE_NAME_CONVERT=(ORA5.DBPRIM,ORA5.DBSTNDBY)
```

Information on the standby database feature can be found in *Oracle Data Guard Concepts and Administration*.

LOCK_SGA

The `LOCK_SGA` parameter is ignored on Oracle Database for z/OS. Buffers in the SGA are pagefixed during I/O operations, only; otherwise, the SGA on z/OS is pageable.

LOG_ARCHIVE_

The parameters whose names begin with "`LOG_ARCHIVE_`" control Oracle database processing of filled database log files when the database runs in `ARCHIVELOG` mode. These parameters changed in Oracle 8.1 to allow more than two copies of archived logs and to support transmitting logs to a remote standby database. The old parameter forms are still supported in Oracle Database 10*g*, but the new forms must be used to allow for more than two log destinations or to use remote standby logging.

With Oracle Database for z/OS, `LOG_ARCHIVE_DEST` and `LOG_ARCHIVE_DUPLEX_DEST` (the old parameters) or the `LOCATION` component of each `LOG_ARCHIVE_DEST_`*n* parameter are used to specify only the left-hand portion of a full z/OS data set name. This will be the high-level data set name qualifier, or qualifiers, for all logs archived to this destination. The remainder of the data set name is specified using `LOG_ARCHIVE_FORMAT`, which should contain the log thread and sequence number substitution indicators (%T and %S) to ensure that each archived log data set name is unique. The destination string must end with a period, or the format string must begin with one in order to form a proper z/OS data set name. The following example uses the newer parameters:

```
LOG_ARCHIVE_FORMAT = "T%T.S%S.R%R"
LOG_ARCHIVE_MIN_SUCCEED_DEST=1
LOG_ARCHIVE_DEST_1='LOCATION=ORA5.ARCHLOG1. MANDATORY REOPEN=5'
LOG_ARCHIVE_DEST_2='LOCATION=ORA5.ARCHLOG2. MANDATORY REOPEN=5'
LOG_ARCHIVE_DEST_STATE_1=enable
LOG_ARCHIVE_DEST_STATE_2=enable
```

With these example settings, archived logs will have data set names of the form `ORA5.ARCHLOG1.T`*`nnn`*`.S`*`nnn`*`.R`*`nnn`* and `ORA5.ARCHLOG2.T`*`nnn`*`.S`*`nnn`*`.R`*`nnn`* with the *nnn* parts replaced by log thread, sequence numbers, and reset logs ID, respectively.

When archiving to a standby database, use the `SERVICE` keyword to specify a valid net service name from the `tnsnames.ora` file, as in the following example:

```
LOG_ARCHIVE_DEST_4='SERVICE=standby1'
LOG_ARCHIVE_DEST_STATE_4=enable
```

The specified service name must use PROTOCOL=TCP.

SGA_MAX_SIZE

This parameter should not be specified on Oracle Database for z/OS. It will default to the actual SGA size. An incorrect value will cause the database to fail to start.

SPFILE

The SPFILE (server parameter file) parameter refers to the VSAM linear data set managed by the instance. The default name of the SPFILE data set is the following concatenation: &ORAPREFD..&ORASRVN..SPFILE.ORA.

When creating an SPFILE, the text parameter file you specify with the PFILE parameter is read by the database server, not by the program that is issuing the CREATE statement (for example, SQL*Plus). The filespec you give for the PFILE can be a sequential data set, a PDS member, a DD filespec, or the path and name of a file in the POSIX HFS. If you use a DD filespec, the server region JCL must contain the appropriate DD statement and it must specify a sequential data set or a member of a PDS. If you are using an HFS file, the filespec must be unambiguous in order for the server to recognize it as an HFS file. Refer to the *Oracle Database User's Guide for IBM z/OS (OS/390)* for information on ambiguous and unambiguous filespecs.

STANDBY_ARCHIVE_DEST

This parameter is used in conjunction with the standby database availability feature to perform database recovery. It is used along with the LOG_ARCHIVE_FORMAT parameter to generate the fully qualified standby database log filenames, which are then stored in the standby database control file. For more information on the standby database availability feature, refer to *Oracle Data Guard Concepts and Administration*.

With Oracle Database for z/OS, the STANDBY_ARCHIVE_DEST parameter is used to specify only the left-hand portion of a full z/OS data set name. This will be the high-level data set name qualifier, or qualifiers, for all logs archived to this destination. The remainder of the data set name is specified using LOG_ARCHIVE_FORMAT, which should contain the log thread, sequence number, and reset logs ID substitution indicators (%T, %S, and %R) to ensure that each archived log data set name is unique. The destination string must end with a period, or the format string must begin with one in order to form a proper z/OS data set name. The following is an example:

```
LOG_ARCHIVE_FORMAT = "T%T.S%S.R%R"
```

```
STANDBY_ARCHIVE_DEST='ORA5.ARCHLOG.'
```

TRACEFILE_IDENTIFIER

This parameter is not used to form part of the trace data set name, but with `ALTER SESSION`, will cause a new trace data set to be allocated.

Pre-Allocating Database Files

You can allow the Oracle server to create the database VSAM clusters during `CREATE DATABASE` processing, or you can pre-allocate them yourself using z/OS IDCAMS. If you choose to use z/OS IDCAMS, now is the time to do it. Considerations for defining database files are covered in Chapter 4, "Defining z/OS Data Sets for the Oracle Database". If you are going to let the server create any files, then be sure to provide creation parameters for the associated file types via the server `ORA$FPS` DD statement as described in Chapter 4, "Defining z/OS Data Sets for the Oracle Database".

Configuring z/OS Security

If you plan to use any of the SAF-based security features discussed in Chapter 9, "Security Considerations", you may want to configure them now. OSDI subsystem command security affects the service `START` command that you will issue to start the database service address spaces. Both OSDI bind security and protection of the database `SYSDBA` or `SYSOPER` privilege affect the connection that you will make to startup Oracle Database for z/OS and create the database.

Even if you do not utilize any of these features, you must ensure that the database service address space is authorized to access resources (such as data sets) that might be protected by default in your system. Refer to Chapter 9 for information on database service interaction with z/OS security.

Configuring for Shared Servers

Shared servers on Oracle Database for z/OS are supported by the generic listener running in a UNIX System Services shell environment. For information on configuring the generic listener for shared servers, refer to Chapter 8, "Oracle Net". For more information on shared servers, refer to the *Oracle Net Services Administrator's Guid*e.

Creating the Database

When the OSDI service has been defined, all required JCL procedures and parameter files have been prepared, and any database files that you want pre-allocated have been defined, then you are ready to create the Oracle database.

First, start the OSDI service with an OSDI `START` command. This is described in Chapter 5, "Operating a Database Service", and in Appendix A, "OSDI Subsystem Command Reference". Check the z/OS system log to make sure that the service starts successfully, as indicated by message MIR0002I. If problems occur, the service address space(s) will terminate. In this case, correct the problems and issue another OSDI `START` command.

After the database service is started, you can use SQL*Plus or a similar tool to perform Oracle startup and database creation. z/OS considerations for running SQL*Plus are covered in the *Oracle Database User's Guide for IBM z/OS (OS/390).*

Before issuing `STARTUP` or `CREATE DATABASE`, the tool must connect to the running database service. The form of `CONNECT` command that is used when starting the database is special and does not specify an Oracle userid:

```
CONNECT / AS SYSDBA
```

This command connects the tool as Oracle userid SYS. If you configured SAF-based OSDI bind authorization as discussed in Chapter 9, "Security Considerations", the z/OS userid associated with the SQL*Plus session must be authorized for the `UBIND` resource associated with the database service. If you configured SAF-based protection of the SYSDBA and SYSOPER privileges as discussed in Chapter 9, "Security Considerations", the userid must also be authorized for the `DBA` or `OPER` resource associated with the database service.

When SQL*Plus processes the `CONNECT`, it must determine the target database service. On z/OS, you can specify the target service in several ways, all based on the SID that is specified in `DEFINE SERVICE`. These methods of specification are described in detail in the *Oracle Database User's Guide for IBM z/OS (OS/390)*. In our examples, we have used the dummy `ORA@`*`sid`* DD statement to specify the target service.

If `CONNECT` fails, the subsequent `STARTUP` and `CREATE` statements will fail as well. This does not affect the running service. You can simply correct the `CONNECT` problem and try again.

After a successful `CONNECT`, you are ready to issue `STARTUP` with the `NOMOUNT` option, which causes the server instance to initialize without trying to mount and open a database. Before doing so, you have the option to create an `SPFILE` (a server parameter file), which is a VSAM LDS containing a binary representation of your Oracle database instance `init.ora` file parameters. If you do not create the `SPFILE`, you will need to supply the `PFILE` parameter on your `STARTUP` command, specifying the text file containing your instance parameters. For more information on `SPFILE` and `PFILE` processing, refer to the section"Oracle Initialization Parameter Considerations" on page 3-17 and to the *Oracle Database Administrator's Guide* and *Oracle Database Reference*.

If `STARTUP` fails (because of an error in your `init.ora` file, for example), the subsequent `CREATE` will fail as well. In addition to any messages displayed by the tool, messages might be in the instance alert log (usually, though not necessarily, a `SYSOUT` file). In general, you can correct errors and retry the whole sequence without stopping and restarting the service.

After `STARTUP` is successful, you can issue `CREATE DATABASE`. This is a SQL statement and is therefore documented in *Oracle Database SQL Reference*. (`CONNECT` and `STARTUP` are commands, not SQL statements, and are described in the SQL*Plus tool documentation.) There are few z/OS-specific considerations for `CREATE DATABASE`. The syntax conventions for file names that are specified to the server are covered in Chapter 4, "Defining z/OS Data Sets for the Oracle Database".

If you pre-allocated any of the database files, be sure to specify `REUSE` in the appropriate clauses. For files that you want the server to create, omit REUSE and, except for the control files, specify `SIZE`. Oracle automatically calculates `SIZE` for the control files based on other parameters in `CREATE DATABASE`. With Oracle Database for z/OS, you can take advantage of the Oracle Managed Files feature and allow the server to generate names for data sets created during `CREATE DATABASE`. Pay careful attention to the `CREATE DATABASE` parameters whose names begin with "`MAX`", because these parameters specify limits that cannot be changed later without recreating the control file.

As `CREATE DATABASE` is processing in the server, it reads the Oracle bootstrap SQL file, usually referred to as `sql.bsq`. On z/OS, this file must be specified via a `SQLBSQ` DD statement in the service JCL procedure as described in "Database Region JCL" in Chapter 3, "Configuring a Database Service and Creating a New Database". This file is read only during processing of a `CREATE DATABASE`, and the DD statement can therefore be removed from the procedure if desired. It does no harm to leave it in, however.

After `CREATE DATABASE`, you must run Oracle-supplied SQL scripts to build the Oracle dictionary, stored PL/SQL procedures, and related structures. Although this can be done immediately after `CREATE DATABASE` (in the same tool session), we chose to run it separately in our examples. `CREATE DATABASE` is therefore the last statement in this part of the example.

Our example of z/OS database creation is presented as a batch job that uses SQL*Plus. We have pre-allocated four log files and a single file for the SYSTEM tablespace, but we are going to let the server create the control files (whose names are specified in the init.ora parameter file, which is supplied via the INITORA DD in this job). The SID of the target instance is ORA1. The /NOLOG in the SQL*Plus PARM prevents SQL*Plus prompting for an Oracle userid and password. The batch job is shown in the following example:

```
//*--------------------------------------------------------------*
//*                                                              *
//*  JOB DESCRIPTION: Create Oracle database                     *
//*                                                              *
//*--------------------------------------------------------------*
//*
//SQLPLUS  EXEC PGM=SQLPLUS,PARM='/NOLOG',REGION=0M
//STEPLIB  DD  DISP=SHR,DSN=ORACLE.V10G.CMDLOAD
//ORA$LIB  DD  DISP=SHR,DSN=ORACLE.V10G.MESG
//*  REQUIRES //ORA@SID  DD DUMMY STATEMENT (ORACLE INSTANCE).
//ORA@ORA1 DD DUMMY
//SQLBSQ   DD  DISP=SHR,DSN=ORACLE.V10G.SQL(SQLBSQ)
//ORA$ENV  DD  DISP=SHR,DSN=ORACLE.ORA1.PARMLIB(ORA1ENV)
//SQLLOGIN DD DUMMY
//SYSIN    DD *
SET ECHO ON
CONNECT / as SYSDBA
@'//ORACLE.ORA1.PARMLIB(CREATE)'
```

The job may run for a while depending on the number and sizes of the files that are specified. Oracle formats all of the primary space for all control, log, and data files.

The PARMLIB CREATE member contains the following:

```
STARTUP PFILE='ORACLE.ORA1.PARMLIB(INITORA)' NOMOUNT
CREATE DATABASE ORA1
  MAXDATAFILES 255
  MAXLOGFILES  255
  CONTROLFILE REUSE
  LOGFILE 'ORACLE.ORA1.LOG1' REUSE,
          'ORACLE.ORA1.LOG2' REUSE
  DATAFILE 'ORACLE.ORA1.SYSTEM.DB1' REUSE,
           'ORACLE.ORA1.SYSTEM.DB2' REUSE
  SYSAUX DATAFILE 'ORACLE.ORA1.SYSAUX.DB1' REUSE
  UNDO TABLESPACE "UNDOTBS"
       DATAFILE 'ORACLE.ORA1.SYSTEM.UNDO.DB1',
                'ORACLE.ORA1.SYSTEM.UNDO.DB2'
  DEFAULT TABLESPACE USERS
          DATAFILE 'ORACLE.ORA1.USER.DB1',
                   'ORACLE.ORA1.USER.DB2'
  DEFAULT TEMPORARY TABLESPACE "TEMP"
          TEMPFILE 'ORACLE.ORA1.TEMP.DB1'
  CHARACTER SET "WE8EBCDIC1047";
```

Populating the SYSTEM Tablespace

After the `CREATE DATABASE` completes successfully, your database is mounted and open. Before you can create application tables, users, and so forth, you must create the Oracle dictionary tables, stored procedures, and other internal structures. SQL scripts for this purpose are placed in a PDS during Oracle installation. As with database creation, you can use SQL*Plus to process these scripts against your new database.

For a new database, members `CATALOG` and `CATPROC` from the SQL PDS must be run, in that order. Both can be done in a single tool execution as in the following example. Because these scripts contain embedded references to other scripts that are members of the same PDS, you must use FNA to control file name processing in the tool. (FNA is explained in the *Oracle Database User's Guide for IBM z/OS (OS/390)*.) As with our previous example, the target database service is identified by an `ORA@`*`sid`* DD statement. We are using the same `CONNECT` statement as the create job.

```
//*-----------------------------------------------------------------*
//*                                                                 *
//*  JOB DESCRIPTION: run catalog and catproc                       *
//*                                                                 *
//*-----------------------------------------------------------------*
//*
//SQLPLUS  EXEC PGM=SQLPLUS,PARM='/NOLOG',REGION=0M
//STEPLIB  DD  DISP=SHR,DSN=ORACLE.V10G.CMDLOAD
//ORA$LIB  DD  DISP=SHR,DSN=ORACLE.V10G.MESG
//*  REQUIRES //ORA@SID  DD DUMMY STATEMENT (ORACLE INSTANCE).
//ORA@ORA1 DD DUMMY
//ORA$ENV  DD  DISP=SHR,DSN=ORACLE.ORA1.PARMLIB(ORA1ENV)
//SQLLOGIN DD DUMMY
//SYSIN    DD *
SET ECHO ON
CONNECT / as SYSDBA
@/oracle/v10102/rdbms/admin/catalog.sql
@/oracle/v10102/rdbms/admin/catproc.sql
/*
```

The "@" symbols that are used in SYSIN in the example above are SQL*Plus shorthand notation for reading an alternate input file of commands or SQL statements. Following each "@" is a member name in the SQL PDS. The FNA controls (ORA$FNA DD statement) are used to notify the tool that the names following the "@" are members in the PDS that is identified by the LIB DD statement.

These two scripts are large and will therefore run for quite a while. Because the scripts create and load data into tables, log file data is generated as they execute. If you specified ARCHIVELOG in your CREATE DATABASE statement (as in our example), logs may fill and require archiving while CATALOG and CATPROC run. You may want to avoid this complication because you are not likely to attempt a recovery of a brand new database: in the event of problems, both CATALOG and CATPROC can simply be rerun. To avoid log archiving during catalog creation, specify NOARCHIVELOG in your CREATE DATABASE, and then use ALTER DATABASE to switch to ARCHIVELOG mode later.

Be sure to check the output of the catalog build job carefully. Because the scripts are designed to be rerunnable, they contain DROP statements that produce errors the first time they are run. These errors are normal. Other errors must be investigated and resolved to complete initialization of the database.

Note: The database session which executes the catalog build requires up to 96M of session-private memory. If you have limited session memory to less than 96M with the database region MAX_SESSION_MEM parameter, catalog build may fail with an ORA-04030 error, an LE/370 U4088 ABEND, or other errors.

The following example initializes the Java VM:

```
//*--------------------------------------------------------------*
//*                                                              *
//*  JOB DESCRIPTION:  Initialize Java VM.                       *
//*                                                              *
//*--------------------------------------------------------------*
//*
//SQLPLUS  EXEC PGM=SQLPLUS,PARM='/NOLOG',REGION=0M
//STEPLIB  DD  DISP=SHR,DSN=ORACLE.V10G.CMDLOAD
//ORA$LIB  DD  DISP=SHR,DSN=ORACLE.V10G.MESG
//*  REQUIRES //ORA@SID  DD DUMMY STATEMENT (ORACLE INSTANCE).
//ORA@ORA1 DD DUMMY
//ORA$ENV  DD  DISP=SHR,DSN=ORACLE.ORA1.PARMLIB(ORA1ENV)
```

```
//SQLLOGIN DD DUMMY
//SYSIN    DD *
SET ECHO ON
CONNECT / as SYSDBA
@/oracle/v10102/javavm/install/initjvm.sql
/*
```

Depending on other Oracle products or features that you may use, you may need to run additional scripts against your new database to enable those products or features. Refer to the product-specific documentation for more information.

After CATALOG and CATPROC have run, your database is ready for use. If you created your database with NOARCHIVELOG to avoid archiving logs during catalog build, then return it to ARCHIVELOG mode by shutting down Oracle, starting it back up with the MOUNT and EXCLUSIVE options, and issuing ALTER DATABASE ARCHIVELOG then ALTER DATABASE OPEN. This also is a good time to change the passwords of the Oracle userids SYS and SYSTEM, which are set up with default passwords during CREATE DATABASE. (This should be done regardless of whether you are changing ARCHIVELOG mode.) Both of these actions are shown in the following example.

```
//*--------------------------------------------------------------*
//*                                                              *
//*  JOB DESCRIPTION: Change database passwords                  *
//*  The passwords SECRET* need tailoring before running this job  *
//*--------------------------------------------------------------*
//*
//SQLPLUS  EXEC PGM=SQLPLUS,PARM='/NOLOG',REGION=0M
//STEPLIB  DD  DISP=SHR,DSN=ORACLE.V10G.CMDLOAD
//ORA$LIB  DD  DISP=SHR,DSN=ORACLE.V10G.MESG
//*  REQUIRES //ORA@SID  DD DUMMY STATEMENT (ORACLE INSTANCE).
//ORA@ORA1 DD DUMMY
//ORA$ENV  DD  DISP=SHR,DSN=ORACLE.ORA1.PARMLIB(ORA1ENV)
//SQLLOGIN DD DUMMY
//SYSIN    DD *
SET ECHO ON
CONNECT / as SYSDBA
startup pfile='ORACLE.V10G.PARMLIB(INITORA)' mount
alter database open;
alter user sys identified by SECRET1;
alter user system identified by SECRET2;
alter user dbsnmp identified by SECRET3;
```

You can now proceed with creating ids for Oracle users, adding tablespaces and tables for applications, and so forth.

Creating an Oracle Database Instance

After installing Oracle Database for z/OS, you use the Oracle Universal Installer configuration utility to create one or more Oracle database instances. The configuration utility creates `INSTLIB` and `PARMLIB` libraries customized to your environment. Each database instance will have its own set of `INSTLIB` and `PARMLIB` libraries. These libraries contain the JCL procedures needed to create the database instance.

Before using the configuration utility, determine the following information for the database instance to be created:

- Subsystem name of the database
- Names of the JCL procedures for the subsystem
- Database SID
- High-level qualifier for all PDS and VSAM data sets
- Volumes on which PDS and VSAM data sets will reside

This information should be determined in advance, because it is used multiple times in creating the database members and can be more complicated to change later. For all other information, you can accept the defaults and change them manually, later.

The following steps provide guidelines for creating a database instance using the configuration utility. They assume that you have already installed Oracle Database for z/OS, performed the APF authorization, and put the necessary files in the linklist. If not, you will need to complete those tasks before continuing. For more information, refer to the *Oracle Database Installation Guide for IBM z/OS (OS/390)*.

Step 1: Run the Configuration Utility

Start Oracle Universal Installer and select the Oracle z/OS Database and Subsystem Configuration option. This starts the configuration utility which prompts for the following information:

- High-level qualifiers. Specify the high-level qualifiers for the the location of the Oracle executable code and the database. Although you can use one high-level qualifier for both, it is recommended that you use a separate high-level qualifer for each.

  The high-level qualifer for the location of the Oracle executable code (*`oracle_hlq`*). For this high-level qualifier, you should include the version information, for example `ORACLE.V10G`.

The high-level qualifier for the Oracle database (*db_hlq*) is used to identify the PDS data set files for the database instance. This high-level qualifer should be labeled with the database name or a similar name, for example ORACLE.ORA1. It is recommended that you do not include version information in the database high-level qualifier, as this is likely to change over time.

- INSTLIB and PARMLIB libraries. Specify where to create the new INSTLIB and PARMLIB libraries. This can be done using IBM SMS (Storage Management Subsystem) or by manually specifying a volume and unit.
- Subsystem Definition Parameters. Specify the subsystem name, database SID, net SID, port on which the Oracle database should listen for remote connection attempts,and names of the JCL procedures for the database instance.
- Server Parameters. Specify the basic OSDI parameter file definitions for hte database service.
- ORA$FPS Control File Definitions. Specify the information to create an ORA$FPS file. This file is used to create various database files of a particular size. You can enter new values for your installation or enter default values and later modify the ORA$FPS file that is created. The VOLSER (volume serial number) used for control files and database files is also used in the DEFINE JCL procedure.

The configuration utility generates two PDS data set files for the INSTLIB and PARMLIB libraries, *db_hlq*.INSTLIB and *db_hlq*.PARMLIB.

The INSTLIB library contains all the sample JCL required to create the database instance and perform simple tasks like starting and stopping the database.

The PARMLIB library contains all the required parameter files needed to create the database instance.

Step 2: Customize JCL Procedures and Parameter Files

The degree to which the information provided in Step 1 is accurate will determine how much the JCL procedures and parameter files need to be modified in order to create the database instance.

JCL Procedures

All JCL procedures (or batch jobs) need to be reviewed carefully to ensure that they are valid. The JOBCARD job provided is only a default used by Oracle and will need to be tailored to your environment.

The batch jobs can be divided into the following categories:

- Two sample `PROCS` that define the Oracle address space and the Net address space, and a procedure to copy them into a system installation library.

  The `PROCS` are named in step 1 of the configuration process, and are cross-referenced to the OSDI parameter file. Each procedure must be defined to RACF as a started task and associated with a user. The following are examples of RACF commands required to perform these tasks:

  The batch job used to copy the two `PROCS` into a system `PROCLIB` library job is called `COPYPROC` and must be customized to point to the system `PROCLIB` library.
- Sample `PROCs` which are used on a system-wide basis for Oracle utilities. These usually begin with the name ORA*, and should be customized and copied into system `PROCLIB` libraries before the new database goes into production. Initially, they are not needed for the basic installation.
- JCL procedures required to define, start and set up the database. These all begin with `SQL`*`sidn`* where *`sid`* is the SID name chosen for the instance, and *`n`* is a number from one to eight. These procedures perform specific tasks related to creating the database. The other batch jobs associated with this are called `DEFINE` and `STRTSRVC`. The `DEFINE` job is used to define the database files and the `STRTSRVC` job is used to define and activate the subsystem. Another set of `PROCS` provide basic functions like starting and stopping the database and starting the subsystem services.

Parameter Files

The parameter files are located in *`db_hlq`*`.PARMLIB`. They are the core definition files for the database. You should review these files for accuracy before you create the database instance. The parameter files can be divided into the following categories:

- OSDI parameter files. Of this group of files, the the subsystem definition file is the core file. This file is called only by the OSDI subsystem name. This file has a major impact the other JCL batch jobs and parameter files and should be modified with care. The following is an example of the subsystem definition file:

```
INIT (ORASSI,SSN1)
DEF SVG SSID(SSN1) DESC('OSDI Oracle 10G Subsystem - SSN1')

DEF SRV ORAS10 PROC(ORA1S10) TYPE(ORA) MAXAS(1) -
 DESC('Oracle V10G RDBMS Service') -
 SID(ORA1) PARM('ORACLE.ORA1.PARMLIB(SSN1PARM)')
```

```
DEF SRV ORAN10 PROC(ORA1N10) TYPE(NET) -
 DESC('Oracle V10G Net Service') -
 SID(ORAN) PARM('HPNS PORT(1501) ENCLAVE(SESS)')

SHOW SERVICEGROUP LONG

START  ORAS10
START  ORAN10
```

The file related to this is called *sid*PARM. This contains all the OSDI specific parameters. These parameters are documented in detail in this guide. Most default parameters are acceptable for most basic installations. The following is an example fo this file:

```
* SSN1 OSDI SUBSYSTEM PARAMETER FILE.
* USED BY SUBSYSTEM SSN1 SERVICE ORAS10 PROC ORA1S10
* LOAD MODULE TO USE.
SERVER_LOADMOD(ORACLE)
* NUMBER OF ADDRESS SPACES TO START. VALUES ARE 1-256
INIT_ADR_SPACES(1)
* MAXIMUM NUMBER OF SESSIONS ALLOWED FOR THIS ADDRESS SPACE
MAX_SESSIONS(500)
* MAX MEMORY ALLOWED PER SESS. VALUES ARE NNNN {K|M}.
MAX_SESSION_MEM(100M)
* INITIAL STACK SIZE. VALUES ARE NNNN {K|M}.
INIT_STACK_SIZE(256K)
* TURN ON SMF RECORDING. VALUES 0 AND 128 THROUGH 255.
SMF_STAT_RECNO(0)
* EXTERNAL AUTHENTICATION
LOGON_AUTH(NONE)
* STORAGE CUSHION. VALUES ARE NNNN {K|M}.
REGION_MEM_RESERVE(10M)
* DATABASE DATASET NAME PREFIX.
DSN_PREFIX_DB(ORACLE)
* TRACE DATASET NAME PREFIX.
TRACE_DSNAME(ORACLE.ORA1.TRACE.&ORASESST..T&HHMMSS)
* ALERT DATASET NAME PREFIX.
ALERT_DSNAME(ORACLE.ORA1.ALERT.&ORASESST..T&HHMMSS)
* MINIMUM SIZE OF ALERT FILE. VALUES ARE NNNN {K|M}.
ALERT_MIN(10M)
* MAXIMUM SIZE OF ALERT FILE. VALUES ARE NNNN {K|M}.
ALERT_MAX(20M)
```

The third file in this section is called SUBSYS. It provides the command which needs to be issued in order to define and activeate the subsystem definition file.

- The *sid*FPS parameter file contains all the default FPS parameters for creating the database files. Remove any which are not required. It contains the six most common data set types. If this is defined correctly, thendefining additional database files becomes easier. The minimum that is recommended is DFLT. Refer to this guide for details about what to code for this parameter file. An example is as follows

```
* Default parameters
FILE_GROUP(DFLT)
  RECALL(NONE)
  MOUNT(NO)
  DEFAULT_SPACE(10000 10000)
  UNIT(SYSDA)
```

- Instance-specific files

  These are files which relate to the Oracle instance itself. They are the INITORA which is the standard init*sid*.ora file found on other platforms with default parameters for z/OS. This file should be modified to match any installation requirements. The installation utility does not ask many questions about tailoring this parameter file . It prompts only for the SID and number of processes because most parameters can be allowed to take the default for most installations.

  The CREATE file which contains the SQL required to create a new Oracle database instance. This file along with the DEFINE JCL defines the basic layout of an Oracle instance. The type of instance it builds is a small, multi-purpose instance and follows the basic rules of Oracle database design. For example, it is highly recommended that there are at least two control files in any instance, the system data goes into the system tablespace and all user data goes elsewhere.

  The *sid*ENV contains the environment variables required for the Oracle database. The most important is the ORACLE_HOME variable which should point to the ORACLE_HOME for the installation. The *sid*FNA is not required and is primarily used for compatibility reasons. It provides default naming conventions for SQL files.

  Two other files which can be used are the TNSNAMES file and the SQLNET file. The TNSNAMES file provides the default TNSNAMES entries for users to access this instance both through cross memory and through TCP/IP.

Step 3: Copy the Subsystem PROCs to a System PROCLIB

When the `COPYPROC` JCL has been customized correctly and the PROC names defined for the database are valid, run the `COPYPROC` JCL. It copies the two subsystem `PROC`s into a system `PROCLIB` library.

Step 4: Define Initial Database Files

This step defines the basic instance VSAM files requried for an Oracle database. Review the `DEFINE` JCL procedure. Verify that the `IDCAMS` definitions are valid for your site and that the volumes specificed have the necessary space on them. Then run this procedure and make certain that all database files are defined correctly. If any modification to names, types sizes of files is requried then the corresponding change needs to be done in the `CREATE` member of `PARMLIB`.

Step 5: Define and Start OSDI Services

This can be done by running the `STRTSRVC` member. This uses the `SETSSI` command to define the OSDI subsystem. If the user is not able to issue this command then the define service needs to be done by a system programmer. A sample of the command to issue is provided by the `SUBSYS` file in the `PARMLIB`.

At the end of this step, you should have an RDBMS and a Net subsystem defined and active.

Step 6: Create the Database Instance (SQL*sid*1)

There are two ways the database instance can be created. This guide will only consider the method involving JCL and batch jobs. The second method is to use SQL commands from the UNIX System Services shell environment. This method is similar to a manual installationon the UNIX platform.

For batch, a database administrator needs to run the job `SQL`*`sid`*`1`. This job takes the `CREATE` member of `PARMLIB` and creates an `ORACLE` instance in the address space started in the previous step. Using a UNIX System Services shell, a database administrator would need to run SQL*Plus and invoke the commands specified in the `CREATE` member.

Step 7: Run Catalog and Catproc SQL Scripts (SQL*sid*2).

In this step, you run the `catalog` and `catproc` SQL scripts for the database instance. This is done by submitting batch job `SQL`*`sid`*`2`. The job is fairly slow and can take a significant amount of time. The output from this step needs to be carefully reviewed to make certain there are no Oracle error messages. The SQL files used for this step reside in a UNIX System Services shell under in the

`ORACLE_HOME/rdbms/admin` directory. In a UNIX System Services shell, the same step can be implemented by changing the directory to `ORACLE_HOME/rdbms/admin`, connecting as `SYSDBA` and then running the `catalog` and `catproc` scripts.

This step must complete successfully because it will have an impact on every aspect of the database instance. No other jobs should be run against the database instance while this job is running.

Step 8: Initialize Java (SQL*sid*3)

This step can be done by running `SQL`*`sid`*`3`. In a UNIX System Services shell, you would change the working directory to `javavm/admin`, connect as `SYSDBA` and run the `initjvm.sql` script. This step creates the Java virtual machine in the Oracle instance. The output from this step should be reviewed carefully to make certain there are no errors.

After this step has completed it is recommended that the instance be shut down and restarted. This is the final step for any installation.

The following steps must be implemented while the instance is mounted and open.

Step 9: Load SQL*Plus Help Data (SQL*sid*4)

This job loads the SQL*Plus help data into the database. The database must be running when this job is run. You can omit this job if you do not want online help or if you have limited database space.

You might receive messages indicating that a synonym, table, or view does not exist. These messages are normal and do not indicate an error condition. Examine the output for other error messages.

In order to run this step, submit batch job `SQL`*`sid`*`4` or change the working directory to `ORACLE_HOME/sqlplus/admin` and run `pupbld.sql` and `hlpbld.sql` scripts.

Step 10: Create Sample Users (SQL*sid*5)

This step is optional and creates the Oracle sample user `SCOTT`. This job creates the user and demonstration tables referrred to in the Oracle product documentation: the `SCOTT` userid and the `EMP`, `DEPT`, `SALGRADE`, and `BONUS` tables. The database must be running when this job is run. You can omit this job if you do not want the demonstration tables installed or if you have limited database space. To install this job, run `SQL`*`sid`*`5` or, from a UNIX System Services shell, change the working directory to `ORACLE_HOME/rdbms/admin`, run SQL*Plus and connect as `SYSTEM`, and run the `utlsampl.sql` script.

You might receive messages indicating that a synonym, table, or view does not exist. These messages are normal and do not indicate an error condition. Examine the output for other error messages.

Step 11: Install Oracle Intermedia Text (SQL*sid*6)

This job sets up the database for use with Oracle Text. It creates the user `CTXSYS` and the database objects that are required by Oracle Text. User `CTXSYS` is created with the password `CTXSYS`, which you can change after this job completes by using the SQL command `ALTER USER`. The database must be running when this job is run. This job is optional and should be run only by installations requiring Intermedia text.

To install Intermedia Text run the batch job `SQL`*`sid`*`6`, or, from a z/OS UNIX System Services shell, change the working directory to `ctx/admin`, run SQL*Plus and connect as `SYSDBA`, and run the `catctx.sql` script.

Step 12: Install Spatial Data Option (SQL*sid*7)

This job sets up the spatial data option. It should only be run by installations requiring this feature. In order to set up the spatial data option, run `SQL`*`sid`*`7` or, from a UNIX System Services shell, change the working directory to `md/admin`, run SQL*Plus and connect as `SYSDBA`, and run the `catmd.sql` script.

Step 13: Turn on Archiving for the instance (SQL*sid*8)

It is recommended that you turn on archiving for any database in which the data is of critical importance. It greatly enhances recoverability of the database. Without it, the only real database recovery that can be performed is to restore from the last cold backup. In order to turn on archiving, modify the `init.ora` parameter file to uncomment and customize the `ARCHIVE_LOG` parameters, then shut down the instance and run `SQL`*`sid`*`8`.

Step 14: Change All Oracle Default Passwords

This is critical for security purposes. For installation purposes, Oracle uses a number of default passwords. These need to be altered once the installation has completed so that there is no security exposure.

4 Defining z/OS Data Sets for the Oracle Database

This chapter describes z/OS-specific considerations for the files used by an Oracle database. In general, what Oracle product literature refers to as "files" will be data sets in your z/OS system. Some Oracle product features access files that are part of the hierarchical file system of UNIX System Services rather than z/OS data sets. However, the files that make up an Oracle database are all z/OS data sets.

Refer to *Oracle Database Concepts* and the *Oracle Database Administrator's Guide* to learn about the uses and relationships among Oracle databases, tablespaces, and operating system files. Refer to the *Oracle Database User's Guide for IBM z/OS (OS/390)* for a discussion of the interaction between Oracle database tools and z/OS data sets.

The following topics are included:

Oracle Database Files

A number of files are used by an Oracle database instance. Some of these files are not an intrinsic part of the database. These include input parameter files, trace or diagnostic log files, and other similar files. Most of these will be sequential data sets, PDS members, or JES spool (`SYSOUT`) files on your z/OS system.

All of the files comprising an Oracle database are VSAM LDS clusters on z/OS. (For compatibility with past releases, VSAM Entry-Sequenced Data Sets with a control interval size of 4 KB are also accepted.) The Oracle server can create these data sets for you by internally invoking the z/OS `IDCAMS` utility and passing it `DEFINE CLUSTER` commands. This is done automatically during processing of SQL statements that create or add to the database structure, including `CREATE DATABASE`, `CREATE TABLESPACE`, and certain `ALTER`...`ADD` statements.

In most cases, it also is possible to pre-allocate Oracle database files by invoking `IDCAMS` yourself and issuing your own `DEFINE CLUSTER` commands prior to issuing SQL statements that add the files to the database. The main advantage to pre-allocation is that it gives you more control over physical data set placement.

If you omit file specifications on a `CREATE DATABASE` request, Oracle normally creates a minimal set of files using default data set names discussed in the following sections. This practice is not recommended for production databases. An optional facility called Oracle Managed Files (OMF) provides a different default naming scheme that is aimed at production database use. OMF is discussed in "Oracle Managed Files on z/OS" on page 4-15.

Control File

The control file is a relatively small file used to record Oracle database instance control information. Control file availability is critical for overall database availability, and the Oracle database will therefore maintain multiple mirror-image copies of the control file if told to do so. Oracle Corporation recommends using at least two control file copies for any production database. The control file copies should be on separate physical storage devices and, if possible, on separate I/O paths in order to minimize the risk of losing both (or all) of them to media or path failure. You specify the data set name(s) of the control files in the `init.ora` parameter file that you supply to the Oracle database `STARTUP` command.

If you do not specify a name for the control file and you are not using OMF, the following default file name will be used:

```
"&ORAPREFD..&ORASRVN..DFLCNTL.DBF"
```

The system symbols `&ORAPREFD` and `&ORASRVN` are discussed in Appendix C, "Oracle Database for z/OS System Symbols".

Database Files

The database files contain all the database data, both application tables, indexes and the like, and the Oracle database server's internal dictionary objects. Each tablespace in your database contains at least one database file. The `SYSTEM` tablespace, where the Oracle database keeps its internal dictionary structures and stored PL/SQL procedures, is created automatically when you issue `CREATE DATABASE`.

The names of one or more files to be used for the `SYSTEM` tablespace are supplied in the `CREATE DATABASE` statement or generated by the Oracle server using Oracle Managed Files. Normally, you will not store application data in the `SYSTEM` tablespace: after your database is created, you will create one or more additional tablespaces for application data. The `CREATE TABLESPACE` statement supplies the names of files to be used when adding a new tablespace, or the names can be generated by the Oracle server using Oracle Managed Files.

If you do not specify names for the database files and you are not using OMF, the following default file names will be used:

`"&ORAPREFD..&ORASRVN..DFLDBS1.DBF"` (default database file)

`"&ORAPREFD..&ORASRVN..DFLTEMP.DBF"` (default temporary tablespace file)

`"&ORAPREFD..&ORASRVN..DFLDBU1.DBF"` (default undo tablespace file)

The system symbols `&ORAPREF`D and `&ORASRVN` are discussed in Appendix C, "Oracle Database for z/OS System Symbols".

Redo Log Files

The redo log files record changes to database data (both application data and internal control data) and are critical for restart and recovery. An Oracle database instance must have at least two redo log files. Redo data is written to the log more or less sequentially from the beginning of the file to the end. Like control files, the Oracle server can write multiple mirror images of a log to reduce the risk of data loss due to media failure.

The term "log file group" refers to one or more mirror-image copies of a given log. When a log file (or group) is filled, the Oracle server automatically switches log writing to the next available log file or group. The frequency of this switching is a function of the size of the redo log files and the amount of database update activity.

The data set names of the log files can be supplied in the `CREATE DATABASE` statement, or generated by the Oracle server using Oracle Managed Files. In addition, log files can be added to a database later, using `ALTER DATABASE...ADD LOGFILE`.

If you do not specify names for the redo log files and you are not using OMF, the following default file names will be used:

`"&ORAPREFD..&ORASRVN..DFLLOG1.DBF"` (default redo log file # 1)

`"&ORAPREFD..&ORASRVN..DFLLOG2.DBF"` (default redo log file # 2)

The system symbols `&ORAPREFD` and `&ORASRVN` are discussed in Appendix C, "Oracle Database for z/OS System Symbols".

Archive Log

When you operate the database in `ARCHIVELOG` mode (normally the case for all production databases), the Oracle server copies filled logs to another (newly-created) file in order to make the filled log available for reuse. The new file that is created by such an operation is called an archive log. The data set names of archive logs are generated using a pattern that you specify in your `init.ora` file parameters. This name includes components that are distinct for each log, ensuring a unique data set name for each archive. This is one case where file pre-allocation is not possible: archive logs are always created by the Oracle server via internal `IDCAMS` call. Archive logs usually are required (read) during recovery, after restoring database files from backups. For more information, refer to Chapter 6, "Database Backup and Recovery".

Tablespaces and z/OS Space Management

When you create a tablespace, its initial size is the sum of the sizes of all datafiles you specify on the `CREATE TABLESPACE` statement. These are the actual sizes of the data sets, if you are using preallocated data sets or reusing existing files, or the amounts specified with the `SIZE` keyword if the server is creating the files. In the latter case, the Oracle server uses only the `SIZE` amount, and some unused space (overallocation) may be present due to VSAM allocation rounding. The unused space is never more than one cylinder per file, but you should be aware of this rounding when planning your Oracle server disk space needs.

Enlarging a Tablespace

A tablespace can be enlarged in two ways: by extending existing files of the tablespace or by adding new files. Existing files can be extended both automatically, on demand as data is added, and explicitly with a DDL SQL statement. Automatic file extension is enabled when the `AUTOEXTEND` clause is included in the file specification of `CREATE` or `ALTER TABLESPACE`. Automatic file extension is attempted when a database insert or update requires more free space than is currently available in the tablespace. This can be triggered by any user session; no special privileges are required. Explicit extension is requested with an `ALTER DATABASE DATAFILE...RESIZE` statement using SQL*Plus or a similar interface, and must be done from a session with `ALTER DATABASE` authority (normally a database administrator).

Adding new files to a tablespace is only done manually, by issuing an `ALTER TABLESPACE ADD DATAFILE` statement using SQL*Plus or a similar interface.

There are several z/OS-specific considerations with file extension and `AUTOEXTEND`. When a tablespace consists of multiple datafiles, all specified with `AUTOEXTEND`, you cannot control which data set the Oracle server extends in a given situation. Second, the amount by which to extend a file is governed by the `NEXT` amount from the `AUTOEXTEND` clause, never by a secondary space quantity in the data set's ICF catalog entry. (Secondary space is not included on server-generated `DEFINE CLUSTER` commands. Secondary space that you specify on the `DEFINE CLUSTER` for a preallocated database file is ignored except for the influence it exerts on CA size and rounding.)

VSAM Space Allocation Rounding

VSAM space allocation always rounds an allocation amount to a Control Area (CA) multiple, and that can distort the way the `AUTOEXTEND NEXT` amount is handled. CA size is not specified explicitly but is derived from other `DEFINE CLUSTER` parameters: it is the lesser of the primary space quantity, the secondary space quantity (if any), and the device cylinder size, but never less than one whole track. When the Oracle server issues `DEFINE CLUSTER` it does not include a secondary space quantity, so CA size is one cylinder unless your `SIZE` is very small (less than a cylinder). You can control CA size on preallocated database files by specifying a secondary space amount that is less than a cylinder; although the secondary amount is not used by the Oracle server, it does determine CA size as described here.

The Oracle server is not aware of VSAM space rounding, so if your `NEXT` amount is less than one CA, the space added by CA rounding is not used. It does get used, however, when the file is extended again. In some cases, file extension is satisfied entirely with existing allocated but unused space, without invoking VSAM secondary allocation. Be aware of VSAM space rounding behavior when choosing a `NEXT` amount for a datafile.

Extending Files Accessed from Multiple Server Address Spaces

Another consideration with file extension concerns Oracle database servers that are configured to run in multiple address spaces (where the `DEFINE SERVICE` command specifies a `MAXAS` number greater than one). The z/OS-specific logic for file extension requires establishing two concurrent "opens" to a file that is accessed from multiple server address spaces. This in turn requires that the VSAM cluster be defined with cross-region shareoption 3. The cross-system shareoption is irrelevant.

Using the AUTOEXTEND Clause

If you preallocate a data set for use as a tablespace file and you want to specify `AUTOEXTEND` for it, you must specify SHR(3,3) in your `DEFINE CLUSTER` command. If you let the server create the file for you, you must use the `SHAREOPTION` file management parameter to specify that cross-region shareoption 3 be included on the server's `DEFINE CLUSTER` command. (Server file management parameters are discussed in "Server File Management Parameters" on page 4-8.) This is required only for Oracle servers whose `MAXAS` is greater than one, and is so even if only one address space is actually started. If your Oracle server is defined with `MAXAS(1)` (which is the default), `AUTOEXTEND` processing does not require `SHR(3,3)`.

Bigfile Tablespaces

A bigfile tablespace is a tablespace with a single, but very large datafile (an ultra large datafile). Traditional smallfile tablespaces, in contrast, can contain multiple datafiles, but the files cannot be as large. On z/OS, datafiles associated with a smallfile tablespace are limited to 4 GB; datafiles associated with bigfile tablespaces may be greater than 4 GB and can span multiple volumes.

On z/OS, ultra large datafiles are implemented as extended-format data sets with the Extended Addressabiltiy attribute set. Extended-format data sets must be system managed and are requested through the SMS data class `DSNTYPE=EXT` parameter and subparameter `R` (required). ISMF can be used to create such an SMS data class.

Oracle recommends that you pre-allocate ultra large datafiles. For more information, refer to the section "Special Considerations for Ultra Large Datafiles" on page 4-14.

There are implicit methods for specifying an SMS data class at datafile creation time, whether pre-allocated or otherwise, which may be used at the installation's discretion. It is also possible to configure a file management parameter file group definition with the `DATACLASS` keyword to explicitly provide the data class for datafiles created by the database server. In this case, all datafiles governed by that file group would be allocated as extended-format data sets, including datafiles associated with a smallfile tablespace.

Server File Name Syntax

When specifying data sets or files to an Oracle database server on z/OS (in `init.ora` file parameters, SQL statements, PL/SQL package calls, or other) you use an extended version of the filespec syntax used by IBM C/C++ and LE. The IBM syntax encompasses data sets, DD statements (which can designate a variety of data set or file types), and POSIX HFS files. An extension provided by Oracle provides an easy way to specify spool (`SYSOUT`) data sets.

This section discusses a few filespec considerations that are specific to the Oracle database server. For more information on supported filespecs and syntax, refer to the *Oracle Database User's Guide for IBM z/OS (OS/390)*. This guide also covers ambiguous filespecs which are those that cannot be distinguished as data set or HFS files by syntax alone. Such filespecs are interpreted based on the LE POSIX setting. If POSIX is ON, the filespec is treated as an HFS file. If POSIX is OFF it is treated as a data set. For example, the string "`my.sql`" is ambiguous: it could refer to an HFS file `my.sql` in the current working directory or it could refer to a data set named `MY.SQL`, perhaps with an implied userid prefix.

With some server features, interpretation of ambiguous filespecs is specific to the feature rather than to LE POSIX. For example, all of the files comprising the database store (such as control files, log files, and tablespace files) are VSAM LDS on z/OS. An ambiguous filespec supplied as such a filename, for example in `CREATE TABLESPACE`, is always processed as a data set filespec.

Conversely, some server features are limited to processing POSIX HFS files only and so always interpret ambiguous filespecs as HFS. This is true of the `java.io` package in Oracle Java.

Finally, some server features can access both data sets and HFS files. This is true of server `PFILE` processing (in `CREATE SPFILE...FROM PFILE`) and the

UTL_FILE PL/SQL package. With these components, ambiguous filespecs are interpreted in the server as data set references.

Unambiguous syntax must be used when an HFS file is to be accessed. In contrast to client-side processing, there is no implied high-level prefix on data set names in the Oracle database server, even when a data set filespec does not use enclosing apostrophes. With database files, the system symbol &ORAPREFD is often used as the first qualifier of a database file name; it is translated to a value supplied as a database region parameter.

Similar to data sets, server references to HFS files are never path-relative, implying a current working directory. All server HFS references must use a complete path and file name.

A number of server file access features used by end users and applications use the DIRECTORY database object as a security control mechanism. On z/OS, a DIRECTORY object can specify either an HFS path or the high-level part of a data set name.

Server File Management Parameters

To create a new database or add files to an existing database, you issue a SQL CREATE or ALTER statement to the Oracle server. This statement allows you to specify whether a file is pre-allocated or, if not, the size with which the new file should be created by the server. There is no provision in Oracle SQL for supplying additional z/OS-specific parameters for creating the associated data set. Instead, you can supply file management parameters, using the ORA$FPS DD statement in the database service JCL procedure, to control most of the parameters in the IDCAMS DEFINE CLUSTER command that the Oracle server issues to create a new file.

File management parameters are also used for certain non-VSAM files, including the sequential backup data sets created and read with an RMAN External Data Mover (EDM). In the EDM case the parameters are read and used by the ORAEDM program running in a separate address space instead of the Oracle server.

In the Oracle server, the ORA$FPS file management parameters are meaningful mainly for files for which the Oracle server issues the IDCAMS DEFINE CLUSTER command. As discussed in the section "Oracle Database Files" on page 4-2, you may have the option of pre-allocating database files by issuing your own DEFINE CLUSTER command. The DEFINE CLUSTER command is discussed in "Pre-allocating Database Files" on page 4-13. This command gives you complete control over most of the DEFINE parameters, which may be desirable when creating the permanent parts of a production database.

Oracle server archive log files cannot be pre-allocated, however. Assuming that you are running your database instance in ARCHIVELOG mode, these files are created by the Oracle server whenever an online log fills and is archived. They are created using a generated data set name whose pattern you control with `init.ora` file parameters. Because these files must be created by the server, the file management parameters for the DBAL group are critical for proper operation of the database.

Server file management parameters are read during service startup. You can change the parameters and cause the server to reread them without stopping and restarting the service, by using a MODIFY command. This is discussed in the section "Other Database Service Commands".

File Group Names

File management parameters are organized by file group, with each group having a distinct four-character name. The current file group names are as follows:

- DBAL - database archive log file (VSAM)
- DBAT - database alert log file (non-VSAM)
- DBAU - database autobackup file (VSAM)
- DBBA - database archive log backup piece (RMAN backup to disk) (VSAM)
- DBBI - database incremental backup piece (RMAN backup to disk) (VSAM)
- DBBK - database datafile backup piece (RMAN backup to disk) (VSAM)
- DBCH - database change tracking file (created for ALTER DATABASE ENABLE BLOCK CHANGE TRACKING) (VSAM)
- DBCP - database datafile copy (VSAM)
- DBCT - database control file (VSAM)
- DBDB - database datafile (one that is part of a tablespace) (VSAM)
- DBDR - database disaster recovery configuration file (VSAM)
- DBOL - database redo log file (VSAM)
- DBOS - database transferred file (output of the DBMS_FILE_TRANSFER package) (VSAM)
- DBPM - database text parameter file (including PFILE) (non-VSAM)

- DBSP - database server parameter file (`SPFILE`) (VSAM)
- DBST - database binary trace file (VSAM)
- DBTR - database text trace file (non-VSAM)
- DBTS - database tempfile (VSAM)
- NTPM - network text parameter file (non-VSAM)
- NTTR - network text trace and log files (non-VSAM)
- P*nnn* - RMAN EDM backup file (non-VSAM; *nnn* is `POOL` number)

In addition to the above, the file group name `DFLT` can be specified to supply default parameters. Default parameters are used for any group that you do not specify explicitly.

Special Considerations for External Data Mover

The External Data Mover (EDM), discussed in Chapter 6, "Database Backup and Recovery", executes in a separate address space from the Oracle database server and supports database file backup and restore operations under control of the RMAN utility. EDM uses file management parameters to control the creation of non-VSAM sequential backup data sets on disk or tape.

In this case, the `ORA$FPS` DD statement in the EDM JCL procedure supplies the parameters. The file groups for EDM backups all have names of the form P*nnn* where *nnn* is the storage pool number from an RMAN `BACKUP` request. Certain parameters are specific to EDM and are ignored when specified for file groups used by the Oracle database server.

File Management Parameters and Syntax

The `ORA$FPS` parameter file contains file group definitions which are specified using *`keyword(value)`* syntax. Each definition must start with the keyword `FILE_GROUP(`*`name`*`)` and continues until the next `FILE_GROUP` (name) keyword is encountered. Comments must start with an asterisk (*) and can begin in any column as long as comments (that are on the same line as keywords) are separated from the last keyword by at least one blank.

Keywords can be coded one per line or strung together on the same line separated by at least one blank, but a *`keyword(value)`* pair cannot be split across two lines. No defaults are defined for the parameters. If a keyword is not coded, then it will not be used on the `DEFINE CLUSTER` or dynamic allocation that is used to create data sets in the associated group. The only parameter that can be overridden or supplied from the SQL command line is SPACE, which the Oracle server supports via the SIZE keyword in SQL statements that specify files. The default file group (DFLT) supplies parameters for any file group that is completely omitted from the file management parameters. Keywords are described in the following list:

Keyword	Description
BUFNO(*nnn*)	Specifies the number of I/O buffers to be allocated by EDM for use during conventional (non-proxy) backup and restore operations, where *nnn* is a decimal number from 1 to 255. Each buffer is equal in size to the block size of the associated backup data set. The buffers are allocated above the 16MB line. BUFNO(3) is the default.
CREATE_MODELDSN(*dsn*)	Specifies a data set name to use as a MODEL in IDCAMS DEFINE CLUSTER commands. This value is mutually exclusive with the SMS class name parameters. CREATE_MODELDSN can be abbreviated MODEL.
DATACLAS(*classname*)	Specifies an SMS data class name to be specified on DEFINE CLUSTER or dynamic allocation requests to create new data sets. DATACLAS can be abbreviated DATACL.
DEFAULT_SPACE(*primary secondary*)	Specifies default primary and secondary space quantities for a data set that is being created. The primary quantity applies only in situations where the Oracle server has not indicated the desired file size. The secondary quantity is optional and is meaningful only for non-VSAM data sets created by the server, such as trace files. It is ignored for the VSAM LDS clusters comprising the database. Both values must be numbers and are expressed in kilobyte (1024-byte) units. DEFAULT_SPACE can be abbreviated SPA.
DSS_COMPRESS(YES|NO)	Specifies whether DFSMSdss should compress data sets dumped during a proxy backup. This parameter is applicable to proxy backup only; if specified for conventional backup, it is ignored. YES indicates that DFSMSdss will compress dumped data sets. NO indicates that DFSMSdss will not compress dumped data sets. The default is NO.
EDM_TAPEDSN(*dsname*)	Specifies the data set name to be used for allocation purposes during a proxy backup or restore operation. The data set is not cataloged but it is placed in the tape header. System symbols may be used to ensure uniqueness, thus preventing enqueue contention during concurrent proxy operations. This parameter is applicable to proxy backup only; if specified for conventional backup, it is ignored. The default value is ORACLE.EDMBKUP.A*xxxx*.D&&LYYMMDD..T&&LHHMMSS , where *xxxx* is the address space ID of the EDM.

Keyword	Description
`EXPIRATION_DATE (`*yyyyddd*`|`*yyddd*`)`	Specifies a data set expiration date to be used for allocation during EDM backup data set creation or restoration, where *yyyyddd* is a four-digit year and three-digit day of the year, and *yyddd* is a two-digit year and three-digit day of the year. `EXPIRATION_DATE` can be abbreviated `EXPDT`.
`FILE_GROUP(`*name*`)`	Specifies the file group to which the file management parameters belong, where *name* is one of the allowed 4-letter file group names. This ends any in-progress file group definition and begins a new one. `FILE_GROUP` can be abbreviated `FILE`.
`MGMTCLAS(`*classname*`)`	Specifies an SMS management class name to be specified on `DEFINE CLUSTER` or dynamic allocation requests to create new data sets. `MGMTCLAS` can be abbreviated `MGMTCL`.
`RECALL(ALL|NONE)`	Specifies whether migrated data sets may be recalled during backup data set allocation for an EDM restore operation. ALL indicates that the recall of migrated data sets is allowed during backup data set allocation. `NONE` indicates the recall of migrated data sets is not allowed during backup data set allocation. The default is `RECALL(NONE).`
`SHAREOPTION(`*n*`)`	Specifies the VSAM cross-region shareoption to be used on `DEFINE CLUSTER`. `SHAREOPTION(1)` is the default. It must be specified as `SHAREOPTION(3)` if the associated data set is to use the `AUTOEXTEND` feature in a server configured to run in multiple address spaces. See the discussion of `AUTOEXTEND` and z/OS space management under "Tablespaces and z/OS Space Management" on page 4-4. `SHAREOPTION` can be abbreviated SHR.
`STORCLAS(`*classname*`)`	Specifies an SMS storage class name to be specified on `DEFINE CLUSTER` or dynamic allocation requests to create new data sets. `STORCLAS` can be abbreviated `STORCL`.
`UNIT(`*unitname*`)`	Specifies an allocation unit name to use in dynamic allocation requests that create new non-VSAM data sets. This parameter is intended for future use when non-VSAM files use file management parameters.

Keyword	Description
VOLUMES(*volser*)	Specifies a volume serial number to use in IDCAMS DEFINE CLUSTER commands for VSAM data sets or in dynamic allocation requests that create non-VSAM data sets. EDM uses this parameter for conventional (non-proxy) backups, only. Only a single volume serial can be specified. Because of this limitation, it is recommended that you use storage management class parameters instead of explicit volumes. VOLUMES can be abbreviated VOL.
VOLUME_COUNT(*nnn*)	Specifies a volume count to be used for allocation during EDM backup data set creation, where nnn is a decimal number between 1 and 255. This parameter is normally used when the backup data set is to reside on tape. VOLUME_COUNT can be abbreviated COUNT.

Storage Management Parameter Example

The following is an example of a storage management parameter:

```
* Oracle server file management parameters
* Tablespace data files
FILE_GROUP(DBDB)
DEFAULT_SPACE(10000 )              * a comment
CREATE_MODELDSN(ORBL.ORAV8.DB1)
* Archive logs
FILE_GROUP(DBAL)
DEFAULT_SPACE(11100   9000)
DATACLAS(OSDIDC2) MGMTCLAS(OSDIMC2)  * 2 keywords on one line
* Default for groups not specified
FILE_GROUP(DFLT)
DEFAULT_SPACE(10000   5000)
UNIT(SYSDA) VOL(TEMP01)
```

Pre-allocating Database Files

If you choose to pre-allocate Oracle server control, log, or database files, then you execute the z/OS IDCAMS utility, and you supply one or more DEFINE CLUSTER commands. Alternatively, DEFINE CLUSTER can be issued directly in a TSO session. For details on the DEFINE CLUSTER command, refer to the IBM document *DFSMS Access Method Services for Catalogs*. This section discusses DEFINE CLUSTER requirements specific to Oracle database files.

You can give an Oracle database file any data set name that conforms to your installation's naming standards and/or security requirements. You will specify this name to the Oracle server later (in a SQL statement or, in the case of control files, in the `init.ora` parameter file) using the file name syntax discussed earlier in this chapter.

Note: Oracle Corporation recommends using a consistent set of qualifiers for the left-hand portion of all data set names associated with a given database. Certain Oracle database features, particularly the standby database features, are usable only when all data sets comprising the database share a common set of leftmost data set name qualifiers.

The amount of space to allocate to a file depends on how the file is used and on your requirements. Refer to the *Oracle Database Administrator's Guide* for discussion of database file sizing for each type of file. The IDCAMS `DEFINE` command can specify space in any of several different units. Choose the unit that is easiest for you. The Oracle server has no preference for space allocation units. Space can be specified in tracks, cylinders, megabytes, kilobytes, or records. Any secondary space quantity in your `DEFINE` is ignored by the Oracle server.

When you specify the pre-allocated file to the Oracle server in a SQL `CREATE` or `ALTER` statement, you must omit the `SIZE` keyword and specify `REUSE`, indicating that the file already exists. Except in the case of control files, the Oracle server will use all of the primary space that you pre-allocated. (With control files, the Oracle server uses exactly the amount of space it needs to contain the internal control structures, whose size depends on some of the other parameters of `CREATE DATABASE`. This might be less than the space that you pre-allocated.)

Special Considerations for Ultra Large Datafiles

Oracle recommends that you pre-allocate ultra large datafiles, as in the following example:

```
DEFINE CLUSTER( -
  NAME( ORACLE.V10G.BIGFILE ) -
  LINEAR -
  DATACLASS( ULDCLASS ) -
  MEGABYTES( 6144 ) )
```

After pre-allocating the datafile, the tablespace can be created with the `REUSE` keyword, which causes the pre-allocated datafile to be used. For example:

```
CREATE BIGFILE TABLESPACE bigtbs
          DATAFILE 'oracle.v10g.bigfile'
          REUSE;
```

Special Considerations for VSAM

The `DEFINE` command must specify the `LINEAR` keyword, indicating that a VSAM LDS is being created. An LDS always has a control interval size of 4K and does not contain VSAM logical record structures. `DEFINE` parameters such as `CONTROLINTERVALSIZE` and `RECORDSIZE` are therefore not used. (If space is specified using `RECORDS`, then IDCAMS assumes that each record equates to one 4K CI.) The VSAM `SHAREOPTIONS` default of `SHR(1,3)` is recommended for all database files except when the server is configured for multiple address space (`MAXAS` is greater than one). In this case, if automatic file extension is desired (using the `AUTOEXTEND` clause of the `CREATE/ALTER TABLESPACE` command), `SHR(3,3)` is required.

Depending on the standards of your installation, you may need to specify `VOLUMES` or one or more of the SMS parameters (`STORAGECLASS`, `MANAGEMENTCLASS`, and `DATACLASS`) in the `DEFINE` command.

The following is an example `DEFINE` command for an Oracle database file:

```
DEFINE CLUSTER( -
NAME(VSAM.QUALS.SYSTEM.DBF2)-
LINEAR -
STORAGECLASS(OSCM3A) -
MANAGEMENTCLASS(OMCM3A) -
MEGABYTES(150))
```

No other preparation, loading, or formatting is required before a pre-allocated file is added to the database. When you specify the new file in a SQL statement (such as `ALTER DATABASE` or `CREATE TABLESPACE`), the server will format all of the primary space of the data set. Adding a large file to the database will therefore incur a noticeable delay while formatting is done. (This is true whether files are pre-allocated or created by the server.)

Oracle Managed Files on z/OS

The Oracle Managed Files (OMF) feature of Oracle Database on z/OS simplifies database administration by eliminating the need to specify the names of database files (control, log, and tablespace files) and to delete underlying files when the owning database element is logically dropped.

When you use OMF, you can omit the single-quoted filenames in the `CREATE/ALTER DATABASE` and `CREATE/ALTER TABLESPACE` statements, because the Oracle server generates unique names for each file. When you drop an OMF log file or a tablespace comprising OMF files, the Oracle server deletes the files. In the case of `DROP TABLESPACE`, you can omit the `INCLUDING CONTENTS AND DATAFILES` clause.

To use OMF, you must do the following:

- Specify certain `init.ora` file parameters involved in name generation
- Omit the single-quoted filenames from `CREATE` or `ALTER` SQL statements

The `init.ora` file parameters for OMF are `DB_CREATE_FILE_DEST` and `DB_CREATE_ONLINE_LOG_DEST_`*n*. On z/OS, these parameters supply the left-hand portion of a data set name (high-level qualifier and possibly other qualifiers), and they must end with a period. The OMF parameters can be reset or changed without shutting down, using `ALTER SYSTEM` or `ALTER SESSION`. For z/OS-specific details on the OMF parameters, see "Oracle Initialization Parameter Considerations". For general information about OMF, refer to *Oracle Database Administrator's Guide*.

Even when you specify OMF parameters, you can continue to specify explicit file names in `CREATE` and `ALTER` statements. In fact, it is necessary to do so when you want to use a preallocated file or reuse an existing file.

Oracle tablespace names can be up to 30 characters long. If you want to be able to associate an OMF-created data set with its owning tablespace, then you must use tablespace names that are distinct in the first eight characters.

The right-hand portion of an OMF-generated filename depends on the type of file and includes an encoded timestamp value for uniqueness. The complete data set name format for OMF files is shown in the following example:

```
control files:                     destOMC.Attttttt
log files:                         destOMLnnn.Attttttt
permanent tablespace files:        destOMD.tsn.Attttttt
datafile copy:                     destOMD.tsn.Attttttt
temporary tablespace files:        destOMT.tsn.Attttttt
archive log files:                 destOMA.Tnnn.Attttttt
datafile backup piece:             destOMB.Lnnn.Attttttt
datafile incremental backup piece: destOMB.Lnnn.Attttttt
archive log backup piece:          destOMB.Tnnn.Attttttt
rman backup piece:                 destOMB.Lnnn.Attttttt
rman autobackup piece:          destOMX.xnnnnnnn.Attttttt
block change tracking files:    destOMR.Attttttt
flash back log files:           destOMF.Attttttt
```

In the previous example, the variables are defined as follows:

Variable	Description
dest	is the destination string (_DEST) in the OMF parameter
nnn	is a three-digit log group number
tsn	is up to eight characters of the tablespace name
tttttt	is the encoded timestamp (which looks like a random mix of letters and numerals)
T*nnn*	is the letter "T" followed by a three-digit thread number
L*nnn*	is the letter "L" followed by a three digit incremental level
x	is the letter "P" if the database has an SPFILE or the letter "T" if the database does not have an SPFILE
nnnnnnn	is a seven-byte time stamp

Oracle allows underscores ("_") in a tablespace name, and any that are present are changed to "@" for use in the generated data set name.

Given the 44-character limit on z/OS data set names, the above data set name formats impose a limit of 29 characters on `DB_CREATE_ONLINE_LOG_DEST_`*n* and 23 characters on `DB_CREATE_FILE_DEST` (assuming a tablespace name of eight characters or more).

You can use Oracle-specific and z/OS system symbols in the OMF parameters. The destination string must end with a period after any symbol substitutions have been performed.

SQL statements that exploit OMF are generally the same as their non-OMF counterparts except that single-quoted filenames are missing. REUSE is not recognized for OMF and you can omit SIZE, which defaults to 100M for all types of files.

The following is an example of a `CREATE DATABASE` command that uses OMF for both log files and for the `SYSTEM` tablespace:

```
CREATE DATABASE W1O9
 MAXINSTANCES 1
```

```
MAXDATAFILES 1000
MAXLOGFILES 10
MAXLOGMEMBERS 1
MAXLOGHISTORY 100
LOGFILE SIZE 40M, SIZE 40M, SIZE 40M
DATAFILE SIZE 128M AUTOEXTEND ON NEXT 32M MAXSIZE 256M;
```

The repeated "`SIZE 40M`" results in 3 log files of 40 megabytes each. You can leave the `LOGFILE` clause off completely and the Oracle server will create two log files (or log file groups) of the default 100 MB size. Similarly, the `DATAFILE` clause can be omitted if the 100M size is acceptable and no autoextension is required for the `SYSTEM` tablespace. In fact, if you accept all the defaults, it is possible to specify the `CREATE DATABASE` command as follows:

```
CREATE DATABASE W1O9;
```

Likewise, you can do the same with the `CREATE TABLESPACE` command.

OMF files are distinguished internally by the presence of "`.OMC`", "`.OML`", "`.OMT`", or "`.OMD`" in the data set name. To avoid conflict with OMF, avoid using data set names containing these qualifiers in non-OMF operations.

Copying and Moving Database Files

There are various circumstances in which you might want to copy and move database files. For example, you might want to create a standby database. Two different methods are described in the following examples:

Example 1 This method uses DFSMSdss to copy two tablespaces, system and rollback. It works only for standby when the two systems share DASD.

```
//STEP2    EXEC PGM=ADRDSSU
//SYSPRINT DD SYSOUT=*
//DASD     DD UNIT=SYSDA, SPACE=(CYL, (300,100)),
//            DISP=(NEW,DELETE,DELETE),DSN=&&TMPNAME
//SYSIN    DD *
   COPY TOL(ENQF) CATALOG -
     OUTDDNAME (DASD) -
     DS(INCLUDE( -
      ORAF.V900.SYSTEM.DB1 -
      ORAF.V900.SYSTEM.RBS -
     )) -
     RENAMEU( -
       (ORAF.V900.SYSTEM.DB1,ORBN.V900.SYSTEM.DB1) -
```

```
        (ORAF.V900.SYSTEM.RBS,ORBN.V900.SYSTEM.RBS) -
      )
//*
```

Example 2 This method uses the IDCAMS Export/Import utility to copy the system tablespace. It is useful when the file needs to be transmitted to another system. You can use FTP to send the output file from Export to the remote system and then read it with Import to recreate the file. This method can be used for any database file (control files, log files, and tablespaces).

```
//EXPORTCL EXEC PGM=IDCAMS,REGION=1024K
//SYSPRINT DD SYSOUT=*
//DISKOUT  DD DSN=EXPORT.VSAM,DISP=(,CATLG),
            UNIT=SYSDA,SPACE=(CYL,(50,50),RLSE)
//SYSIN    DD  *
  EXPORT                           -
    ORAF.V900.SYSTEM.DB1           -
     OUTFILE(DISKOUT)             -
     TEMPORARY
/*
```

Now transmit to the remote system the `EXPORT.VSAM` data set, using a method such as FTP.

```
//STEP01 EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=*
//SYSIN    DD *
  IMPORT IDS(EXPORT.VSAM) -
  ODS (ORBN.V900.SYSTEM.DB1) -
  OBJECTS -
  ((ORAF.V900.SYSTEM.DB1              -
    NEWNAME(ORBN.V900.SYSTEM.DB1)     -
   ) -
   (ORAF.V900.SYSTEM.DB1.DATA         -
    NEWNANE(ORBN.V900.SYSTEM.DB1.DATA) -
   ) -
  )
```

5 Operating a Database Service

This chapter describes the z/OS-specific details of how to start and stop an Oracle database instance on z/OS. Use this chapter in conjunction with the information in the *Oracle Database Administrator's Guide* to establish Oracle database server operating practices in your installation.

The following topics are included:

Starting and Stopping the Database Service

As discussed in *Oracle Database Administrator's Guide*, and in the following section, you make an Oracle database instance available for use by issuing the Oracle database `STARTUP` command via Oracle SQL*Plus or a comparable Oracle utility. Before you can do this on z/OS, the OSDI-defined database service must be started. You start a database service using the OSDI `START` command documented in Appendix A, "OSDI Subsystem Command Reference". This will create one or more z/OS address spaces according to the `INIT_ADR_SPACES` value in the server region parameters. These address spaces are what z/OS calls "system address spaces", and they are similar to z/OS started tasks (STCs). Each address space executes the JCL procedure that you specified via the PROC parameter of `DEFINE SERVICE`. The OSDI `START` command can be included in the subsystem parameter file so that the service is always started during IPL.

After the service is started, additional OSDI `START` commands can be issued to create additional address spaces for the service, up to the `MAXAS` limit that was specified in DEFINE SERVICE. Each additional START command adds one address space to the service. Added address spaces increase the amount of virtual memory available for database application sessions. Address spaces can be added before or after the Oracle database STARTUP command is issued. No Oracle database-specific action (such as the STARTUP command) is needed when adding address spaces.

Other than stopping the database service, which terminates all of the service address spaces, there is no way to reduce the number of address spaces of a running service. Database service address spaces cannot be stopped individually.

When you perform an Oracle database shutdown (by issuing the SHUTDOWN command via Oracle SQL*Plus), the associated service address spaces continue to run. You can startup and shutdown an Oracle database instance as many times as you want using the same set of service address spaces. The only situations that dictate stopping the service (terminating its address spaces) are to do the following:

- Effect a change to one of the service parameter files that is read only at service start: ORA$ENV, or the main service parameters (data set specified using the OSDI PARM string)
- Effect a fix or upgrade to software modules (Oracle software, or IBM software that is fetched into the address space such as LE/370)
- Resolve a problem that has rendered the address spaces unusable in some way

With this in mind, it may be best to think of the database service address spaces as more or less permanent fixtures. In fact, the OSDI START command can be included in the subsystem parameter file so that the service is always started during OSDI subsystem initialization (normally during system IPL). This will help to ensure that the service is always ready and available for Oracle database startup processing.

You can stop a database service with the OSDI STOP command (described in Appendix A, "OSDI Subsystem Command Reference") or the native z/OS STOP (or P) command. Stopping a service terminates all of its address spaces. The command takes effect immediately regardless of the operating state of the associated Oracle database instances. If you stop a database service without first performing an Oracle database shutdown via SQL*Plus, then active client requests may be abnormally terminated. The subsequent Oracle database server startup of a database stopped in this fashion will take longer because of the requirement to read log files and perform recovery for transactions that were in progress at the time of termination.

Oracle Database Instance Startup and Shutdown

After the database service is successfully started, you can issue the Oracle database STARTUP command to make the database instance available to applications. Any of several different Oracle database utilities, including SQL*Plus and Recovery Manager, can be used to issue this command. You can execute the utility on the same z/OS system as the database that you are starting, or you can execute it on a different system, even one that is not z/OS. In the latter case, you must have configured and started the Oracle Net network service, and some special security considerations come into play. These special security considerations are discussed in Chapter 9, "Security Considerations". For simplicity, the balance of this section assumes that you are running SQL*Plus on the same z/OS image as the database instance that you are managing.

Most installations will find it convenient to set up started task procedures or operator-startable jobs for executing Oracle database `STARTUP` and `SHUTDOWN` commands so that these functions are accessible to the system operator. For detailed information on executing SQL*Plus on z/OS, refer to the *Oracle Database User's Guide for IBM z/OS (OS/390)*.

Before the utility can issue `STARTUP`, it must establish a connection to the target instance with a `CONNECT` statement. Special rules apply to this `CONNECT` because it is processed before the Oracle database data dictionary (where Oracle database userid information resides) is open.

Special security processing is performed on z/OS to authenticate the z/OS job or user who makes the connection. This processing uses a SAF-based (`RACROUTE`) test, discussed in Chapter 9, "Security Considerations". If your installation has enabled this processing, then the userid that runs SQL*Plus must have been granted the proper authority; otherwise, `CONNECT / AS SYSDBA` will fail, as will the following `STARTUP` command. Refer to Chapter 9 for more details.

You must ensure that the utility `CONNECT` statement connects to the correct database service. When you are running the utility on z/OS, several different methods exist to indicate the target service for a `CONNECT`, all of which utilize the SID associated with the service. These methods are documented in the *Oracle Database User's Guide for IBM z/OS (OS/390)*. In most of our examples here, we use the `ORA@`*`sid`* DD statement to specify the target service.

When using multiple server address spaces, the `STARTUP` command must be issued from a session that is connected to the first service address space. There is no explicit mechanism for requesting connection to the first address space, but this condition is always met when there are no other users connected to the service, which is normally the case when you are running `STARTUP`.

The following is an example Oracle database startup that has been set up as a z/OS batch job using SQL*Plus. The target service has the SID "ORA1", which has been specified using the ORA@ dummy DD statement in the job. The init.ora parameter file is a member of a PDS. The STARTUP command has defaulted the MOUNT and OPEN options, and the database will therefore be opened and made usable to applications.

```
//ORASTART JOB 1,'Oracle ORA1 Startup'
//PLUS EXEC PGM=SQLPLUS,PARM='/NOLOG'
//STEPLIB  DD DISP=SHR,DSN=ORACLE.V10G.CMDLOAD
//ORA$LIB  DD DISP=SHR,DSN=ORACLE.V10G.MESG
//SYSERR   DD SYSOUT=*
//SYSOUT   DD SYSOUT=*
//SQLLOGIN DD DUMMY
//ORA@ORA1 DD DUMMY
//INITORA  DD DISP=SHR,DSN=ORACLE.ORA1.PARMLIB(ORA1INIT)
//SYSIN    DD *
CONNECT / AS SYSDBA
STARTUP PFILE=/DD/INITORA
EXIT
/*
```

The database is shut down in the normal Oracle database fashion, by connecting with SQL*Plus and issuing the SHUTDOWN command. Shutting down the database does not terminate service address spaces.

Oracle Corporation does not recommend using SHUTDOWN ABORT or STARTUP FORCE with z/OS database instances. These commands attempt to forcibly terminate all processes that are accessing the instance, which usually is not a desirable action. In situations where other forms of Oracle database shutdown do not appear to be working, or when you are unable to connect to the server to issue a shutdown command, the best course of action is to stop and restart the database service, then proceed with STARTUP.

Managing the Alert Log

Each Oracle instance you run produces an alert log, which is a sequential log of text messages pertaining to overall database operations including Oracle startup and shutdown, major database events such as log archiving and tablespace definition, and certain categories of errors. Submitting a copy of your alert log data is required in many problem determination situations. Refer to the *Oracle Database Administrator's Guide* for general information on the alert log.

On z/OS, the alert log is implemented as a sequential data set written with BSAM. It can be a spool (SYSOUT) data set or a DSORG=PS disk data set. When the alert log is a spool data set, you can view and copy an instance's active, in-use alert log data using a facility such as IBM's SDSF. This is not possible when the alert log is a disk data set: IBM BSAM does not support concurrent read and write access to a disk data set from distinct tasks or address spaces. If you use a disk data set for the alert log and you need to view or copy the contents (for example, when investigating a problem), you must cause the current alert log data set to be closed and released (unallocated) by the server.

Closing and Releasing the Alert Log

There are two ways to effect close and release of the alert log. One is to stop the OSDI service hosting the instance (normally after performing an Oracle shutdown). Service termination processing will flush all buffered alert log data to disk and then close and unallocate the data set.

Stopping the OSDI service may be inconvenient in situations where the alert log from a production instance is required. For those circumstances, an alert log "switch" mechanism is provided. When a switch occurs, the current alert log is flushed, closed, and unallocated and a new alert log is allocated and opened. This occurs without affecting application activity in the server or server availability in general.

Two mechanisms are provided for switching the alert log. One is the LOGSWITCH database service command, issued via the z/OS MODIFY system command at a system console or comparable facility. This causes an immediate switch of the alert log, freeing the current log for viewing or copying. Besides on-demand use, installations might consider issuing this command automatically on a periodic basis via a system command scheduling facility, to "spin off" accumulated alert log data on a regular basis such as daily or weekly. The database region parameter ALERT_MIN can be used to set a minimum alert log size below which a normal LOGSWITCH command is ignored. A FORCE option on the LOGSWITCH command can be used to override this minimum. Details on the LOGSWITCH command are in the following section.

You also can configure a database service to switch the alert log automatically when it reaches a given size. The OSDI database region parameter ALERT_MAX, described in Chapter 3, sets a size threshold at which the log is switched automatically. As alert log records are written the number of bytes in each logical record is tallied; when the instance issues an alert log message that would exceed the threshold size, the current log is flushed, closed, and unallocated and a new log is opened (and the tally zeroed) before the new record is written.

The alert log also switches automatically if any errors occur when writing to the current log, including the case where BSAM WRITE incurs a System x37 ABEND due to data set or disk volume space being exhausted. Unlike most other Oracle platforms, the alert log is not automatically closed and opened at Oracle shutdown and startup, respectively.

Regardless of which alert log switch mechanism is used, switches occur without regard to the sequence or interrelationship of messages being issued. Oracle could be issuing multiple related messages at the moment a switch occurs, and those messages may end up "split" across the old and new log data sets. However, alert log messages are never lost or discarded. If allocation or open of a new alert log fails and anything other than the default is being used, an attempt is made to allocate and open a default alert log (see below). If allocation of a default alert log fails, alert log messages are issued to the z/OS system log instead. Once alert log messages start going to the system log they continue to go there until the OSDI service is stopped or a LOGSWITCH command is issued and processed successfully, starting a new alert log file.

By default, the alert log switch allocates a new alert log as a spool data set in the default output class of the server, i.e. SYSOUT=*. You can use the ALERT_DSNAME database region parameter, described in Chapter 3, "Configuring a Database Service and Creating a New Database", to change this. The value given for this parameter can be either a SYSOUT filespec (including output class, form name, and destination) or it can be a data set name filespec, indicating that a sequential disk data set is desired. When you use a data set name filespec, the filespec must include imbedded system symbol references so that each use of the filespec produces a unique data set name.

Typically, z/OS symbols for the current date and time (&LDATE and &LTIME) are used. You can also use OSDI service-specific symbols such as &ORASRVN in the filespec, but OSDI session-level symbols such as &ORASESST can not be used. An example of this parameter with a data set name filespec is as follows:

```
ALERT_DSNAME(&ORAPREFD..&ORASRVN..ALERT.D&LDATE..T&LTIME)
```

An example of the parameter with a SYSOUT filespec is as follows:

```
ADSN(//S:*,AN01,MHQPRT12)
```

Refer to the *Oracle Database User's Guide for IBM z/OS* for additional information on filespecs and filespec syntax.

Allocation details for new alert logs are controlled using server file management parameters in the ORA$FPS DD, described in Chapter 4, "Defining z/OS Data Sets for the Oracle Database". The file group name DBAT supplies particulars for alert log allocation. If your ORA$FPS does not include an entry for group DBAT, the default group (DFLT) is used. File management parameters are meaningful only when the alert log is a disk data set (as opposed to SYSOUT), and only the parameters UNIT, VOLUMES, STORCLAS, DATACLAS, MGMTCLAS, and disk space-related parameters are honored when allocating an alert log data set.

Other Database Service Commands

In addition to the commands described previously, three other commands can be issued to a running database service: DISPLAY SESSION, DUMP SESSION, and REFRESH FPS. These commands use the z/OS MODIFY (or F) command mechanism. To issue these commands, enter the following z/OS system command, where *id* is the service jobname or identifier, and *command* is the command image:

```
F id,command
```

DISPLAY SESSION

The DISPLAY SESSION command displays information for active sessions within a database service. All keywords and values are required.

```
DISPLAY SESSION JOBNAME(job_filter)
```

Abbreviations: D, SESS, JOB

job_filter:

Specify up to eight characters. The value may be suffixed with an '*' or may consist of only an '*' to indicate wildcarding.

Examples:

```
F MYORA8,DISPLAY SESSION JOBNAME(JAOTT3)
```

The above command displays information for sessions initiated by clients with a job name of 'JAOTT3' within the MYORA8 database service.

```
F MYORA8,D SESS JOB(JAOTT*)
```

The above command displays information for sessions initiated by clients with a job name beginning with 'JAOTT' within the MYORA8 database service.

DISPLAY VERSION

The `DISPLAY VERSION` command displays the version and linkedit date of the database service or Net service (OSDI listener) executables.

```
DISPLAY VERSION
```

Abbreviations: `D, VER`

Examples:

```
F ORA1,DISPLAY VERSION
```

The above command displays the version information for the `ORA1` database service.

```
F NET1,DISPLAY VERSION
```

The above command displays the version information for the `NET1` Net service.

DUMP SESSION

The `DUMP SESSION` command creates a machine-readable dump of the address space, or spaces, that are associated with a given session within a database service. All keywords and values are required.

```
DUMP SESSION(sessid) DSN(dataset_name)
```

Abbreviations: `SESS`, `DA`

```
sessid:
```

The Session ID is an identifier of the relevant session in the form of eight hexadecimal characters. Session IDs may be discovered with the `DISPLAY SESSION` command.

```
dataset_name:
```

This identifier is the fully-qualified name of the data set that will contain the dump, and it may be up to 44 characters in length. The data set cannot exist at the time that the `DUMP` command is entered.

Example:

```
F MYORA8,DUMP SESS(00010010) DSN(OSDI.SESSION.DUMP)
```

The above command dumps the address spaces associated with session '00010010' to a data set of the name 'OSDI.SESSION.DUMP' within the MYORA8 database service.

KILL SESSION

The KILL SESSION command terminates active sessions within a database service. All keywords and values are required.

```
KILL SESSION(sessid)
```

Abbreviations: SESS

```
sessid:
```

The Session ID is an identifier of the relevant session in the form of eight hexadecimal characters. Session IDs may be discovered with the DISPLAY SESSION command.

Example:

```
F MYORA,KILL SESSION(00130010)
```

The above command terminates the session with the session id (or SPID) of '00130010' within the MYORA database service.

Network client sessions (sessions on an enclave SRB) may not be terminated immediately if they are executing in the database server. Therefore, it may be necessary to use the SQL statement ALTER SYSTEM SESSION KILL to terminate them.

LOGSWITCH

The LOGSWITCH command is used to cause the current Oracle alert log file to be flushed, closed, and unallocated followed by allocation and open of a new alert log according to the ALERT_DSNAME region parameter.

```
LOGSWITCH [FORCE]
```

Abbreviations: LOGSW

The FORCE option indicates that the switch is to be done regardless of any minimum threshold specified with the ALERT_MIN region parameter. If FORCE is omitted and the total size in data bytes of the current alert log is less than ALERT_MIN, no switch is done and message MIR0612I is issued.

Example:

```
F WFMORA1,LOGSW
```

This command causes an alert log switch if the alert log is at least `ALERT_MIN` bytes in size within the `WFMORA1` database service.

REFRESH FPS

The `REFRESH FPS` command is used to reload the server file management parameters defined "Server File Management Parameters" in Chapter 4, "Defining z/OS Data Sets for the Oracle Database". As during database service startup, the file specified by the `ORA$FPS` DD statement is read to obtain the parameters. Unlike service startup, any error encountered while processing the file contents will cancel the attempted refresh.

```
REFRESH FPS
```

Abbreviations: `REFR`

Example:

F MYORA8,REFR FPS

This command reloads the server file management parameters within the `MYORA8` database service.

6

Database Backup and Recovery

This chapter provides z/OS-specific details on Oracle features and techniques that are used to ensure database availability and correctness.

The following topics are included:

Overview

Before planning or attempting database backup or recovery, you should be familiar with the organization of an Oracle database as described in *Oracle Database Concepts*, with common database administration procedures covered in the *Oracle Database Administrator's Guide*, and with basic Oracle backup and recovery practices documented in the *Oracle Database Backup and Recovery Book Set*.

If you decide to use Oracle Recovery Manager (RMAN) for database backup and recovery, which is highly recommended, refer to the *Oracle Database Recovery Manager Book Set.* Finally, if you plan to implement a standby database, consult the *Oracle Data Guard Concepts and Administration* before reading the related material in this chapter.

Logging and Recovery

Recovery is the process of applying or reapplying database changes to reflect transactions which have been committed. Critical to the recovery process is the

Oracle database redo log (or online log) of database changes that is written by the server as transactions are processed. As each log file or log file group is filled, the Oracle database switches to the next file or group in turn. When you run the database in `NOARCHIVELOG` mode, the process eventually wraps back around to the first log file or group, overlaying the data in that file. Most production Oracle databases run in `ARCHIVELOG` mode, which requires that filled logs be archived (copied to another file, called an archive log) before they can be reused. The archive logs are opened and read by the server in certain recovery situations.

On z/OS, archive logs are VSAM LDS similar to the other database files. Unlike control, database, and online log files, they cannot be pre-allocated and are always created by the server via invocation of the `IDCAMS` utility. This means that the server file parameters governing creation of archive logs (the `DBAL` file group specified in the `ORA$FPS` parameter file) are particularly important: these parameters must be specified so that archive logs can be created readily as needed. For additional information on the `ORA$FPS` parameter file, refer to "Server File Management Parameters" in Chapter 4, "Defining z/OS Data Sets for the Oracle Database".

The naming of archive log data sets is controlled by several `init.ora` file parameters whose z/OS-specific details are discussed in "Oracle Initialization Parameter Considerations" in Chapter 3, "Configuring a Database Service and Creating a New Database". You can specify that multiple archive copies are to be created in order to reduce the chances of losing an archive to media failure. The parameter conventions require you to supply one or more leading (left-hand) data set name prefixes, one for each copy. When you elect to create multiple copies, the respective prefixes must differ to avoid duplicate data set names, for example, "`ORA1.ARCHLOG1.`" and "`ORA1.ARCHLOG2.`". A single additional parameter supplies a template for a data set name suffix that is used on all copies. Normally this suffix contains substitution symbols that are replaced with the logical log thread, sequence number, and reset logs ID, ensuring a unique data set name for each archive.

Backup and Recovery without Recovery Manager

As discussed in the *Oracle Database Backup and Recovery Book Set*, there are several approaches to backing up and restoring all or parts of an Oracle database independent of the Oracle server, using platform-specific utilities of your choice. On z/OS, this can be accomplished using a fast data mover such as IBM DFSMSdss or a comparable product. The Oracle database server requirements are met by any software that can backup and restore a VSAM linear data set without disturbing the contents or organization of the data. The restore can be to a different disk volume and physical location than was originally backed up, but it must create a valid ICF catalog entry for the cluster with the same high-allocated and high-used RBA characteristics as the original data set.

The key to performing independent backup of Oracle database data is making sure that the correct group of VSAM linear data sets is backed up as a logical set. Which data sets should be copied during backup depends on the type of backup you are doing. The various types of backups and the Oracle database files which comprise them are discussed in the *Oracle Database Backup and Recovery Book Set*.

One of the most common forms of independent backup is a tablespace backup, in which all of the files that make up a given tablespace are backed up as a set. This kind of backup can be performed while the Oracle server is running and has the tablespace online and in use. It therefore does not interfere with database availability and has little impact on application performance. When done in this fashion (called a "hot backup"), you must notify the Oracle database server of your actions immediately before and after the backup. This notification is done using SQL `ALTER TABLESPACE` statements. The easiest way to accomplish this is to implement the backup as a 3-step batch job in which the first and last steps execute SQL*Plus to issue the requisite SQL, and the middle step executes the data move. The data move step should be conditioned on successful execution of the first step, as shown in the following example:

```
//ORABKTS1 JOB 1,'Oracle Backup'
//PLUS1    EXEC PGM=SQLPLUS,PARM='/NOLOG',REGION=4M
//STEPLIB  DD DISP=SHR,DSN=ORACLE.V10G.CMDLOAD
//ORA$LIB  DD DISP=SHR,DSN=ORACLE.V10G.MESG
//SYSERR   DD SYSOUT=*
//SYSOUT   DD SYSOUT=*
//ORA@ORA1 DD DUMMY
//SYSIN    DD *
WHENEVER SQLERROR EXIT FAILURE
WHENEVER OSERROR EXIT FAILURE
CONNECT / AS SYSDBA
ALTER TABLESPACE ARREQ1 BEGIN BACKUP;
EXIT
/*
//BACKUP   EXEC PGM=ADRDSSU,COND=(0,NE,PLUS1)
//SYSPRINT DD SYSOUT=*
//BKUPDS   DD DISP=(,CATLG,DELETE),
//  DSN=ORACLE.ORA1.ARREQ1.BKF,
//  UNIT=(TAPE,,DEFER)
//SYSIN    DD *
 DUMP DATASET(INCLUDE(ORACLE.ORA1.ARREQ1.DBF*)) -
```

```
  OUTDD(BKUPDS)
/*
//PLUS2    EXEC PGM=SQLPLUS,PARM='/NOLOG',REGION=4M,COND=EVEN
//STEPLIB  DD DISP=SHR,DSN=ORACLE.V10G.CMDLOAD
//ORA$LIB  DD DISP=SHR,DSN=ORACLE.V10G.MESG
//SYSERR   DD SYSOUT=*
//SYSOUT   DD SYSOUT=*
//ORA@ORA1 DD DUMMY
//SYSIN    DD *
CONNECT / AS SYSDBA
ALTER TABLESPACE ARREQ1 END BACKUP;
EXIT
/*
```

In a recovery situation (following a media failure, for example), the Oracle server must not be accessing the logical structure that is being recovered. In the case of tablespace or individual datafile recovery not involving the `SYSTEM` tablespace, the server can be up and running, but the affected tablespace must be offline. For recovery of the `SYSTEM` tablespace or for full database recovery, the Oracle database instance must be shut down. You should restore all data sets comprising the entity that you are recovering, for example: all tablespace files for a tablespace that is to be recovered. Normally, you will restore using the original data set names of the files. If you change the data set names during restore, you will have to start the Oracle database server without opening the database (using `STARTUP MOUNT`) and issue `ALTER DATABASE RENAME` statements to update the Oracle database control file with the new names.

When all restores are completed and any required renaming has been done, you are ready to begin recovery as described in the *Oracle Database Backup and Recovery Book Set*. During recovery, the Oracle server will open and read all log data that has been written since the backup (that you restored) was taken. Usually this involves both archive logs and online logs. All of the required archive logs must be available during recovery. If you allow archive logs to be migrated by a product such as IBM DFSMShsm, then you must recall them to disk before Oracle attempts to open them.

Recovery Manager on z/OS

The Recovery Manager (RMAN) utility automates and simplifies backup and recovery operations. You provide RMAN an input script that directs backup or recovery processing and RMAN connects to the database being backed up or recovered (the target instance) to execute the script. The actual data movement for backup and restore is done by the target instance, not by the RMAN utility program. For more information on RMAN, refer to the *Oracle Recovery Manager Book Set*.

There are two major categories of RMAN backup and restore processing when RMAN is used with a z/OS target instance. If your RMAN script specifies a DISK channel, backup and restore uses VSAM LDS just like the ones comprising the database. If you use an EDM (External Data Mover) channel, which is unique to z/OS, backup and restore operations are executed in a separate, dynamically-started address space on the target system. With an EDM channel, discussed in the following section, backup and restore operations can use sequential (DSORG=PS) data sets on disk or tape. EDM also supports RMAN proxy backup and restore, invoking the IBM DF/DSS utility program to perform bulk data movement.

External Data Mover

On z/OS, an External Data Mover (EDM) implementation is provided for performing RMAN-initiated data movement and backup maintenance activity. EDM runs as a separate z/OS address space, freeing the Oracle database server address space from system resource conflicts (such as tape device allocation and mounting) and from the processing demands of bulk data movement that are associated with backup and restore operations. Priorities or WLM (Workload Manger) goals of the EDM address spaces can be set as desired, separate from those of Oracle server address spaces.

RMAN `ALLOCATE CHANNEL` commands must specify "EDM0" as the channel type to start an EDM address space. The only other RMAN channel type allowed with a z/OS server is DISK. Allocating a DISK channel does not start an EDM address space. A DISK channel can create and read only VSAM LDS backups on direct access storage devices, and the processing occurs entirely within the Oracle server address space. An EDM channel can process backups that are physical sequential (DSORG=PS) data sets on either tape or direct access storage.

Starting and stopping EDM is done automatically by the Oracle server that hosts an RMAN session. Each RMAN `ALLOCATE CHANNEL` command starts a separate EDM address space on the same z/OS system as the host server. The EDM address spaces are system address spaces (similar to started tasks). Each EDM address space executes the Oracle EDM program ORAEDM.

RMAN backup and restore requests are forwarded from the target instance to EDM for processing. Each EDM can process one set of backups or restores or one proxy operation at a time. If your RMAN script allocates multiple EDM channels, RMAN distributes operations over the multiple EDM address spaces automatically. This concurrency requires asynchronous interaction between the RMAN client and the target instance and is supported only when you use TCP protocol. If you run RMAN on z/OS and connect to the target instance with XM protocol, concurrent operations across multiple EDM channels is not possible.

Before you can use RMAN to perform EDM-type operations on a z/OS system, you must set up a JCL procedure for running EDM address spaces. This procedure must be installed in a system procedure library. The JCL EXEC and DD statement requirements for this procedure are discussed in the section "EDM JCL and Parameters" on page 6-15. You can give the procedure any name that you desire, as long as it does not conflict with another procedure or with any subsystem that is defined to z/OS. You will need to know the name when you code an RMAN `ALLOCATE CHANNEL` command.

You might need to discuss EDM security requirements with your system security administrator. The EDM address spaces must be able to create and open backup data sets using data set names that you specify as part of an RMAN script. This might dictate taking actions to associate a z/OS authorization id with the EDM procedure.

Providing a Snapshot Control File

Certain RMAN synchronization functions require a snapshot control file, which is basically a copy of the Oracle database control file that you create with an `ALTER DATABASE` statement. By default, a z/OS Oracle server expects this file to be specified via a `SNAPCF` DD statement in the server region JCL. Of course, you cannot supply this DD statement until the associated z/OS data set exists, so if you want to rely on the default snapshot identification, then you must go through a 2-step process as described here. (Alternatively, you can use the `SET SNAPSHOT CONTROLFILE NAME` command in an RMAN session to specify the snapshot file by its data set name rather than by a DD name.)

To create the snapshot control file, start up the Oracle server and issue an `ALTER DATABASE` command similar to the following:

```
ALTER DATABASE BACKUP CONTROLFILE TO 'ORACLE.V10G.SNAP.CTL';
```

You can use any valid data set name (inside the single quotes) that conforms to the standards of your installation. In most cases, you will want to use the same high-level qualifiers that you use for other files that are part of this database. After this statement completes successfully, you will have created a snapshot control file with the specified data set name. Note that the file management parameters for the `DBCT` group (specified via the `ORA$FPS` DD statement) affect the way that this data set is created by the server. If you wish, you can pre-allocate the snapshot control file with your own invocation of IDCAMS. If you do so, make it the same size as your existing control file and include the word REUSE after the quoted name in the `ALTER DATABASE` statement.

Now shut down the Oracle database server, stop the associated OSDI service, and add a SNAPCF DD statement to the service JCL procedure similar to the following:

```
//SNAPCF DD DISP=SHR,DSN=ORACLE.V10G.SNAP.CTL
```

Start the OSDI service and startup the Oracle database server. Now your instance has a snapshot control file with the expected default identification.

Identifying Backups

Backups created with RMAN are cataloged z/OS data sets. A separate data set is produced for each copy of each backup piece. You specify the data set name for each backup using the `FORMAT` parameter of the RMAN `BACKUP` command.

Although you can specify a fixed z/OS data set name for `FORMAT`, the parameter is designed to be used as a template: various substitution variables (identified by a "%" prefix) can be included in the `FORMAT` string, and these are replaced with specific values each time the RMAN script executes.

You can use any of the substitution variables shown in Table 6–1 as portions of a template data set name in order to form distinct, valid z/OS data set names:

Table 6–1 FORMAT Parameter Substitution Variables

Variable	Description
%c	specifies the copy number of the backup piece within a set of duplexed backup pieces. If you did not issue the set duplex command, then this variable will be 1. If you issued set duplex, the variable identifies the copy number: 1, 2, 3, or 4.

Table 6–1 (Cont.) FORMAT Parameter Substitution Variables

Variable	Description
%d	specifies the database name. Note: The database name substitution variables (%d and %n) use the database name specified in the CREATE DATABASE command (or in the `init.ora` parameter file). The database name can be up to eight characters long and must start with a letter (or #, $, or @) and only contain letters, numbers, the special characters #, $, @ or the hyphen ('-') in the second and subsequent positions to be valid as a standalone part of a data set name (as in 'ORACLE.BACKUP.%d'). If the database name starts with a number but is otherwise valid and is less than eight characters long, a letter (or #, $, or @) can be placed in the pattern before the %d (as in 'ORACLE.BACKUP.X%d'). Thus, if the %d will be used in the FORMAT string, care should be taken in choosing the database name in the `CREATE DATABASE` command to ensure that it is valid in a z/OS data set name.
%n	specifies the database name, padded on the right with 'X' characters to a total length of eight characters. For example, if `PROD1` is the database name, then PROD1XXX is the padded database name. (See the note under %d, above.)
%p	specifies the backup piece number within the backup set. This value starts at one for each backup set and is incremented by one as each backup piece is created. The number will never exceed four digits (9999 maximum).
%s	specifies the backup set number. This number is a counter in the control file that is incremented for each backup set. The counter value starts at one and is unique for the lifetime of the control file. If you restore a backup control file, then duplicate values can result. Also, `CREATE CONTROLFILE` initializes the counter back to one. If this value exceeds 9,999,999, only the right seven digits of the number will be used. The number will be from one to seven digits long.
%t	specifies the backup set timestamp, which is a four-byte binary unsigned integer value derived as the number of seconds elapsed since a fixed reference time. The combination of %s and %t can be used to form a unique name for the backup set. The unsigned binary integer is formatted into a string in this format: Tssssss. The ssssss is a base-32 representation of the number using the letters A-V and the numbers 0-9. The generated value will always be eight characters in length.
%u	specifies an eight-character name constituted by compressed representations of the backup set number and the time the backup set was created. This will always be eight characters in length and it will start with a letter so it will be valid within a z/OS data set name. Note: In Oracle8*i*, the variable was '.*%u' which generated '.Axxxx.Axxxx'. The '.*%u' is still accepted, but now generates '.xxxxxxxx' (just as if '.%u' were specified).
%D	specifies the current day of the month from the Gregorian Calendar in DD format.

Table 6–1 (Cont.) FORMAT Parameter Substitution Variables

Variable	Description
%F	specifies that dbid (a numeric value that identifies an Oracle database), year, month, day and sequence be combined so that the generated name is unique and repeatable. When %F is used in combination with another string or variable, the generated format is: Iiiiiii.Dyymmdd.Sss. The iiiiiii is a base-32 representation of the dbid using the letters A-V and the numbers 0-9 and will always be seven characters. The yy, mm, and dd are the year, month and day, respectively. The ss is a hexadecimal representation of the sequence. When %F is used alone, it generates a complete data set name: &ORAPREFD..&ORASRVN..OBKCF.Iiiiiiii.Dyymmdd.Sss In addition, when RMAN's control file autobackup feature is used, %F is the default format for the control file backup name.
%M	specifies the current month from the Gregorian Calendar in MM format.
%T	stands for year, month and day in the format YYMMDD. This will always be six characters in length. The first two characters of year are dropped (2008 becomes 08).
%U	specifies a convenient shorthand for &ORAPREFD..&ORASRVN..ORABKUP.%u.P%p.C%c that guarantees uniqueness in generated backup filenames. If you do not specify a format, RMAN uses %U by default. Note: For more information on &ORAPREFD and &ORASRVN, see Appendix C, "Oracle Database for z/OS System Symbols".
%Y	specifies the current year from the Gregorian Calendar in YYYY format.
%%	stands for % (i.e. %%Y is actually the string %Y).

Note: Several of the substitution variables generate numbers only, and should not be used immediately after a period in the `FORMAT` string, or an invalid data set name will result. For example, %s generates a one-digit to seven-digit number. In a `FORMAT` string, a period and a letter (or #, $, or @) have to precede the %s to make it valid in a data set name (for example, 'ORACLE.BACKUP.S%s').

You can also use Oracle Database for z/OS system symbols (identified by a "&" prefix) to form parts of the data set name. This allows a single FORMAT string to produce distinct backup data set names over multiple pieces and copies, and over multiple uses of the same RMAN script. Refer to Appendix C, "Oracle Database for z/OS System Symbols", for more information.

When you code the `FORMAT` parameter, you should combine appropriate high-level data set name qualifiers, %u, other RMAN substitution variables, and Oracle Database for z/OS system symbols to form a complete, meaningful data set name. Both RMAN substitution variables and Oracle Database for z/OS system symbols are translated into their current values as the `BACKUP` command is processed at the server. The Oracle server saves the names of backups in the RMAN catalog or in the database control file, so you do not need to supply, or even know, the names of backups in order to perform an RMAN `RESTORE`.

The following example is a more complex `FORMAT` parameter for Oracle Database for z/OS:

```
FORMAT '&ORAPREFD..BKUP.%d.%u.P%p.C%c'
```

Variables in the previous example are defined, as follows:

Table 6–2 Variable Definitions for Code Example

Variable	Definition
`&ORAPREFD.`	is replaced with the default data set name prefix (a server region parameter). Note that the system symbol, `"&ORAPREFD.",` has the required period terminator.
`%d`	is replaced with the Oracle database name.
`%u`	is replaced by a unique, 8-byte string.
`%p`	is replaced by the backup piece number.
`%c`	is replaced by the backup copy number.

A data set name that is produced from this format specification might look as follows:

```
ORA3ADBF.BKUP.ORA3DB.G6ENPJ03.P3.C1
```

Maintaining EDM Backups

The data set name of a successfully-created RMAN backup is always cataloged in the z/OS system catalog structure. When RMAN is told to delete a backup that is no longer needed, EDM invokes the z/OS IDCAMS utility and passes it a `DELETE` command for the data set. If the backup is a disk data set, then `DELETE` uncatalogs the data set and scratches it from the disk volume. If it is a tape data set, then DELETE only uncatalogs the data set. If the backup was a disk data set, and if it has been migrated by DFSMShsm, then the `DELETE` command logically deletes the migrated copy without recalling it to disk.

When backups are taken directly to tape, the tape volumes must not be reused as long as the backup remains known to RMAN, in other words, as long as the backup is cataloged. The easiest way to ensure this, if you use z/OS tape management software, is to specify "catalog retention" for tape backups. In this case, the tape management system will not recycle the tape volumes as long as they are associated with a cataloged data set name. How you specify catalog retention for a tape data set depends on the tape management software that you are using. If you are using IBM's DFSMSrmm, catalog retention is specified as a retention policy. Such policies can be imposed based on the data set name or on a DFSMS management class. For further information, refer to the IBM DFSMSrmm documentation or to the documentation for the tape management software that your system uses.

Proxy Backup Identification

When you use RMAN proxy backup, one or more whole database files is backed up in a single invocation of the IBM DF/DSS utility running in an EDM address space. RMAN requires that each copied database file in such a backup be logically accessible for for restore processing and for possible deletion.

The backup created by DF/DSS during proxy operation is a single sequential z/OS data set on one or more tape volumes. The name of this data set is not a backup file name; it is controlled instead with the `EDM_TAPEDSN` file management parameter. Although this is the real name of the backup data set (written on the tape's standard label) this data set name is not cataloged in the z/OS system catalog. Instead, each backup piece name is cataloged. Thus, on completion of a proxy backup of ten datafiles, you will have ten different entries in your z/OS catalog pointing to the same set of tape volumes.

During proxy restore, EDM issues a z/OS `LOCATE` macro using a backup piece name to determine the tape device type and volumes required for the DF/DSS restore. When DF/DSS executes, EDM provides DF/DSS control statements to select the particular backup pieces to be restored. Thus, the z/OS catalog entry for any proxy backup piece must persist in order for that backup piece to be usable in a proxy restore.

Conversely, when deleting a logical backup piece during RMAN maintenance processing, the associated z/OS catalog entry is deleted. It is possible for some of the files backed up in a proxy backup to be deleted in this way and others not. As long as any backup pieces remain cataloged the proxy backup should be considered viable and the associated tape volumes not scratched or recycled. When the last backup piece is logically deleted and there are no z/OS catalog entries referring to the tape volumes, the tapes can be reused.

Proxy Backup Restrictions

Proxy backup is designed for backing up entire database files, not for incremental backup. A single proxy operation on OS/390 can copy up to 255 files in a single invocation of DFSMSdss. If your proxy backup operation involves more than 255 files, it has to be divided across multiple RMAN channels or done in multiple RMAN scripts. For information on these techniques, refer to the *Oracle Database Recovery Manager Book Set*.

Only tape devices are supported for proxy backups on z/OS. Disk backups can be made with existing EDM facilities or the Oracle server itself.

Proxy Restore

Whenever it restores a backed up file, RMAN is designed to overwrite the target file. The target file will be overwritten even if you use RMAN's `SET NEWNAME FOR DATAFILE` syntax to rename the restored target file and even if a file with the new name already exists on disk. In contrast, DFSMSdss will refuse to restore a file that is being renamed if a file with the new name already exists on disk. Therefore when using proxy restore with DFSMSdss, you will need to manually delete any files with a new target name before beginning the restore. You will not need to delete any files if you restore to the original file.

For an example that illustrates file management parameters containing file group definitions that might be used for proxy backup and restore operations, refer to "File Management Parameters Example" on page 6-15.

EDM Backup Allocation Parameters

The z/OS-specific parameters for backup data sets are specified indirectly in an RMAN `BACKUP` command using the `POOL` parameter. The actual parameters are supplied in the EDM procedure in a file identified by an `ORA$FPS` DD statement. These parameters are coded the same as those for the database server region, discussed in "Server File Management Parameters" in Chapter 4, "Defining z/OS Data Sets for the Oracle Database". The `ORA$FPS` parameter file that you use with

EDM will contain different file groups with different parameter settings than those used in the database server.

The file group identifier (in the EDM case) consists of the letter "P" followed by a three-digit decimal pool number. For example, if your backup script specifies "POOL 6", EDM searches the ORA$FPS file for parameters associated with file group P006. The POOL parameter defaults to zero, and EDM will therefore search for file group P000 if no POOL is coded in the script.

As with the database server, if the matching group cannot be found, then EDM will use parameters specified for the DFLT (default) group if they are included in the EDM's ORA$FPS file. If no DFLT group exists, then EDM will attempt to create the backup file with only a data set name, disposition, and (possibly) space parameters. This is likely to fail unless your installation uses automatic classification (ACL) logic to set suitable allocation parameters.

Additional details and an example ORA$FPS file are included in the section "EDM JCL and Parameters" below. The full description of ORA$FPS keywords and syntax is in "Database Region JCL" in Chapter 3, "Configuring a Database Service and Creating a New Database".

ALLOCATE CHANNEL Considerations

For a target server on z/OS, the ALLOCATE CHANNEL command must specify the channel TYPE as either DISK or "EDM0". The full quotes must be included when the EDM0 type is used. A DISK channel can be used to create and read only backups or copies that are VSAM LDS. This processing takes place within the target database server address space and does not involve EDM. Allocating a TYPE "EDM0" channel starts an EDM address space that can create and read backups that are physical sequential (DSORG=PS) data sets on disk or tape.

The EDM JCL procedure is identified using the PARMS parameter of ALLOCATE CHANNEL. At minimum, PARMS must specify the one-character to eight-character JCL procedure name for your EDM JCL procedure. The PARMS string can also include other fields that are valid for a z/OS START command, including a job identifier, JOBNAME or SUBSYS keywords, and procedure-specific JCL symbolic keywords. This string is case-sensitive and generally must be supplied in all uppercase letters. If apostrophes are required around a JCL symbolic parameter, then they must be doubled inside the outer apostrophes that are part of RMAN's PARMS parameter syntax.

The following example is an ALLOCATE CHANNEL command for an EDM whose JCL procedure, ORAEDM1, includes a procedure-specific keyword parameter FPS. A job identifier (EDMC1) is also included:

```
ALLOCATE CHANNEL C1 TYPE "EDM0" PARMS 'ORAEDM1.EDMC1,FPS=''FPS1''';
```

The PARMS parameter defaults to an empty string. If you omit PARMS on an EDM channel allocation for a z/OS server, then the command will fail.

If your backup script creates multiple backup pieces or copies, or if a restore script is going to call for multiple pieces or copies, then consider allocating multiple EDM channels to improve concurrency.

Note: The z/OS RMAN client cannot exploit multiple channels for concurrency when a protocol=XM connection to the database is used. Multiple channel concurrency may be achieved by establishing a TCP connection to the database.

Each ALLOCATE command can specify the same EDM procedure name. If you specify a job identifier (or the JOBNAME parameter), Oracle Corporation recommends using a distinct identifier or jobname for each EDM channel in the script.

BACKUP Considerations

The BACKUP command parameters with z/OS-specific considerations are FORMAT and POOL. As discussed earlier, FORMAT must specify a string that will produce a valid, distinct z/OS data set name for each backup piece and copy after RMAN metasymbol and z/OS system symbol substitutions have been performed. FORMAT can also be specified in the ALLOCATE CHANNEL command, in which case the specified string applies to all backups that are created using the associated channel.

The POOL parameter is used to select backup file creation parameters that are supplied via the ORA$FPS DD statement in the EDM JCL procedure. The three-digit pool number (padded on the left with zeroes if necessary) is appended to the letter "P" to form the four-character file group identifier that is looked up in the EDM's ORA$FPS file. If POOL is omitted, it defaults to zero, which means EDM will attempt to locate parameters for file group P000.

Example RMAN Backup Script

```
run {
  allocate channel c1 type "EDM0" parms 'ORAEDM.EDM1';
  allocate channel c2 type "EDM0" parms 'ORAEDM.EDM2';
  backup database format 'ORA1.BKUPDB.%u.P%p.C%c' pool 6;
```

```
    }
```

EDM JCL and Parameters

To use EDM, you must have a JCL procedure in a system procedure library.

For example:

```
//EDMPROC  PROC,FPS=EDMFPS
//EDMPROC  EXEC PGM=ORAEDM,REGION=0M
//STEPLIB   DD DISP=SHR,DSN=ORACLE.V10G.AUTHLOAD
//ORA$FPS   DD DISP=SHR,DSN=ORACLE.ORA1.PARMLIB(&FPS)
//SYSPRINT  DD SYSOUT=*
//PXYPRINT  DD SYSOUT=*
```

where:

Table 6–3 JCL Procedure Descriptions

JCL Procedure	Description
`PGM=ORAEDM`	identifies the External Data Mover program.
`//STEPLIB`	is normally included. It identifies the Oracle AUTHLOAD data set containing ORAEDM.
`//ORA$FPS`	is recommended. It specifies parameters that allow control over the creation, attributes, and processing of backup data sets by EDM. If omitted, internal and system defaults are used.
`//SYSPRINT`	is required. Normally a JES spool data set, it receives informational and error messages generated by EDM.
`//PXYPRINT`	is conditionally required. For an EDM instance to perform proxy operations, this DD must be specified. Normally a JES spool data set, it receives messages generated by DFSMSdss.

File Management Parameters Example

The following example illustrates file management parameters containing file group definitions that might be used for backup and restore operations (both conventional and proxy). Note that Pool 0 is always used for restore operations. This causes FILE_GROUP(P000), or, if not defined, FILE_GROUP(DFLT) to be used:

```
*  Define file group P000 to be used for restore operations.
*  BUFNO() will be honored for only conventional (non-proxy)
*  restore operations; otherwise it is ignored.  This entry
*  works well for both conventional and proxy  restore
*  operations.
*
FILE_GROUP(P000)
BUFNO(20)
RECALL(ALL)
*
* Define file group P001 to use for conventional backup to
*  _disk_ operations.
*
FILE_GROUP(P001)
BUFNO(20)
UNIT(SYSDA)
*
*  Define file group P002 to use for conventional backup to
*  _tape_ operations.
*
FILE_GROUP(P002)
BUFNO(20)
VOLUME_COUNT(255)
UNIT(3490)
*
*  Define file group P003 to be used for proxy backup
*  operations.  These _must_ go to tape.
*
FILE_GROUP(P003)
VOLUME_COUNT(255)
UNIT(3490)
DSS_COMPRESS(YES)
EDM_TAPEDSN(SYSBKUP.ORACLE.D&&LYYMMDD..T&&LHHMMSS)
*
*  Define file group P004 to be used for conventional _and_
*  proxy backup to _tape_ operations. If proxy, BUFNO() will be
*  ignored; if conventional, DSS_COMPRESS() and EDM_TAPEDSN()
*  will be ignored.
*
FILE_GROUP(P004)
BUFNO(20)
VOLUME_COUNT(255)
UNIT(3490)
DSS_COMPRESS(YES)
EDM_TAPEDSN(SYSBKUP.ORACLE.D&&LYYMMDD..T&&LHHMMSS)
```

Since parameters irrelevant to the task at hand are ignored, it is possible to define just one file group definition to cover all flavors of conventional and proxy backup and restore operations. Additionally, if the file group is designated as P000 or

DFLT, as in the following example, lack of a pool specification on a backup command would cause the definition to be used (and restore operations would use it by default):

```
FILE_GROUP(DFLT)
BUFNO(20)
RECALL(ALL)
VOLUME_COUNT(255)
UNIT(TAPE)
DSS_COMPRESS(YES)
EDM_TAPEDSN(SYSBKUP.ORACLE.D&&LYYMMDD..T&&LHHMMSS)
```

7 Oracle Database Administration Utilities

Before using this chapter, you should be familiar with the general considerations for running Oracle tools and utilities on z/OS in the *Oracle Database User's Guide for IBM z/OS*. Common aspects of all Oracle tools and utilities are discussed in detail there, such as basic JCL and other runtime requirements, how to specify connections to an Oracle database server, and how input and output files are specified and processed.

This chapter provides z/OS-specific details on Oracle utilities used primarily for database administration purposes. It also includes general information on administrative features of Oracle client applications, tools, and utilities on z/OS.

The following topics are included:

- Global Environment Variable File
- Recovery Manager (RMAN) on z/OS
- Oracle Password Utility (ORAPWD) on z/OS
- Offline Database Verification Utility (DBV) on z/OS

Global Environment Variable File

Environment variables, discussed in the *Oracle Database User's Guide for IBM z/OS*, are used to control certain aspects of Oracle tools, utilities, and user-written applications. They are named parameters that are supplied locally when the program runs. In POSIX shell environments they can be supplied using native shell mechanisms and in non-POSIX TSO and batch they are supplied using a file identified by an ORA$ENV DD statement or allocation. A client's language, character set, and related locale preferences are among the things controlled by environment variables.

Starting with version 10*g*, Oracle Database on z/OS supports the use of a **global environment variable file**. This is a specific data set or HFS file that is read by all Oracle tools, utilities, and client programs running in a POSIX(OFF) environment. The global environment file is not read by Oracle-accessing CICS TS or IMS TM transactions, nor is it read by Oracle database or gateway instances running on z/OS, nor is it read by Oracle tools and utilities running in a POSIX(ON) environment.

Using a global environment file allows you to set default values for selected environment variables for all non-POSIX clients (other than CICS TS and IMS TM) in your system. This can free users from having to supply a local environment variable setting (for example, an ORA$ENV DD) that is widely used in your installation. Global environment file settings only act as defaults: individual jobs or users can override global settings by specifying the same environment variable with a different value in a local ORA$ENV file.

By default, the global environment file has the following filespec:

```
//'SYS1.ORACLE.ENV'
```

This means it is a sequential data set named SYS1.ORACLE.ENV. During initialization of Oracle tools and utilities, or during the first Oracle interaction in a user-written application, if a data set of this name exists and can be opened, it is read and processed as environment variable settings. By creating a cataloged data set with this name and placing environment variable settings in it, you activate the global environment variable file feature. For more information on filespecs and environment variables, refer to the *Oracle Database User's Guide for IBM z/OS*.

If desired, you can change the filespec that Oracle accesses as the global environment variable file. To do this, you must create a loadable module named ORAENVGL and place it where it will be loaded by all Oracle-accessing programs. Depending on how you have installed Oracle client components, this could be in the Oracle CMDLOAD data set or it could be in a system link list library.

The sole content of the ORAENVGL module is the character string filespec to use for the global environment file, ended by a single zero (X'00') byte. The simplest way to create this module is to assemble and link a small source program, as in the following example:

```
ORAENVGL RSECT
ORAENVGL RMODE ANY
*   Filespec to open for Oracle client global environment
*   variable settings.  Contains single apostrophes, which
*   must be doubled within an assembler character constant.
         DC    C'//''ORACLE.GLOBAL.ENV'''
         DC    X'00'       Required terminator byte
         END   ORAENVGL
```

This example causes applications to try to use the data set ORACLE.GLOBAL.ENV as the global environment file instead of SYS1.ORACLE.ENV.

Whether you need to use a different filespec or not, you should refer to the *Oracle Database User's Guide for IBM z/OS* for more information about environment variable files and environment variables. You may also want to talk to the users who are developing or running Oracle database applications on z/OS to determine if global environment settings might be appropriate in your system.

Recovery Manager (RMAN) on z/OS

RMAN on z/OS supports batch job, TSO, and POSIX shell execution environments. A JCL procedure for batch execution (named ORARMN, by default) is supplied by Oracle and may be installed on your system with the same, or a different, name. In TSO, both CALL and command processor (CP) invocation are supported. For non-POSIX environments, the load module or program object name is RMAN. In a POSIX shell (including OMVS under TSO), use the `rman` command (in lower case) to invoke this utility.

RMAN has some special processing requirements. It must be able to read the `recover.bsq` script during its initialization. In batch and TSO environments this script is the RECOVER member of the SQL data set created during Oracle Database installation. RMAN expects a BSQ DD statement or TSO file allocation that specifies the SQL data set but no member name, as in the following examples:

Batch job or TSO logon procedure:

```
//BSQ DD DISP=SHR,DSN=oracle_hlq.SQL
```

TSO dynamic allocation:

```
  ALLOCATE FILE(BSQ) DA('oracle_hlq.SQL') SHR
```

If you use the ORARMN*xx* procedure, this DD statement is already included. In a POSIX shell this script is `$ORACLE_HOME/rdbms/admin/recover.bsq`. It is also created during installation. Verifying that the ORACLE_HOME environment variable is set correctly will ensure that the script can be read by RMAN.

Depending on how it is used, RMAN may need to connect to as many as three distinct Oracle database instances: one for its catalog (the "catalog instance"), one for the database that is being backed up or recovered (the "target instance"), and, during certain types of point-in-time recovery, an "auxiliary instance" that participates in recovery processing.

The requirement to connect to multiple instances indicates that you cannot rely entirely on one of the singular mechanisms (the ORA@`sid` DD statement or the ORACLE_SID or TWO_TASK environment variables) to specify the instance. You can use one of those mechanisms for any one of your RMAN connections, but the other connection(s), if required, must use a `tnsnames.ora` file or explicit Oracle Net address strings. Oracle Corporation recommends using a `tnsnames.ora` file. Refer to the *Oracle Net Services Book Set* and to Chapter 8, "Oracle Net", for a discussion of this file.

The RMAN CONNECT statements that do not rely on ORA@`sid`, ORACLE_SID, or TWO_TASK will need to supply the `tnsnames.ora` identifier for the instance. In the example batch RMAN job which follows, we have used ORA@`sid` to access the catalog instance at SID 'ORMC' and have used a `tnsnames.ora` identifier to access the target instance at SID 'ORA1'. Only the RMAN CONNECT statements are shown.

```
//ORARMAN  JOB 1,'Oracle Recovery Mgr'
//RMAN     EXEC PGM=RMAN
//STEPLIB  DD DISP=SHR,DSN=ORACLE.V10G.CMDLOAD
//ORA$LIB  DD DISP=SHR,DSN=ORACLE.V10G.MESG
//BSQ      DD DISP=SHR,DSN=ORACLE.V10G.SQL
//SYSERR   DD SYSOUT=*
//SYSOUT   DD SYSOUT=*
//ORA@ORMC DD DUMMY
//TNSNAMES DD *
DBORA1=(DESCRIPTION=(ADDRESS=(PROTOCOL=XM)(SID=ORA1)))
/*
//SYSIN    DD *
connect catalog rman/rman
connect target /@DBORA1
...
/*
```

On z/OS, RMAN sets return code zero if all input statements are processed without error. If any errors occur, a return code of 8 is produced.

Oracle Password Utility (ORAPWD) on z/OS

The Oracle password utility, ORAPWD, is used to initialize a password file that the database server uses to validate certain types of Oracle logon. Usage considerations for a password file (which is optional) are discussed in Chapter 9, "Security Considerations" and in the *Oracle Database Administrator's Guide*.

ORAPWD on z/OS supports batch job, TSO, and POSIX shell execution environments. In TSO, both CALL and command processor (CP) invocation are supported. For the non-POSIX environments, the load module or program object name is ORAPWD. In a POSIX shell (including OMVS under TSO), use the `orapwd` command (in lower case) to invoke this utility.

The password file must be pre-allocated as a VSAM LDS prior to executing ORAPWD. The IDCAMS DEFINE CLUSTER considerations for this file are exactly the same as those for Oracle database files, discussed in Chapter 4, "Defining z/OS Data Sets for the Oracle Database". Refer to the *Oracle Database Administrator's Guide* for information on sizing this file.

All input to ORAPWD comes from the PARM field or command line parameters. When you specify the FILE= parameter to ORAPWD, use only the data set name of the VSAM LDS. Do not include apostrophes or any "//" prefix. The z/OS userid that is associated with the batch job or session must have update authority on the data set. Following is an example batch job that creates the password data set using IDCAMS and then initializes the password data set using ORAPWD.

```
//*-----------------------------------------------------------------*
//*                                                                 *
//*  JOB DESCRIPTION: Define / create ORAPWD file                   *
//*                                                                 *
//*-----------------------------------------------------------------*
//*
//DEFINE  EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=*
//SYSIN    DD *
 DELETE ORACLE.ORA1.ORAPWD
 DEFINE CLUSTER (NAME(ORACLE.ORA1.ORAPWD) LINEAR -
  RECORDS(16))
/*
//ORAPWD  EXEC PGM=ORAPWD,
//  PARM='file=ORACLE.ORA1.ORAPWD password=manager entries=32'
//STEPLIB  DD  DISP=SHR,DSN=ORACLE.V10G.CMDLOAD
//ORA$LIB  DD  DISP=SHR,DSN=ORACLE.V10G.MESG
//ORA@ORA1 DD DUMMY
//SYSIN    DD DUMMY
//*
```

On z/OS, ORAPWD sets return code zero if the file is initialized without errors. If any errors occur, a nonzero return code is produced.

Offline Database Verification Utility (DBV) on z/OS

DBV (Database Verification Utility) examines the physical and logical structure of an offline Oracle database file or a (backup) copy of a database file. General considerations for using DBV are discussed in the *Oracle Database Backup and Recovery Book Set*.

DBV on z/OS supports batch job, TSO, and POSIX shell execution environments. No JCL procedure is supplied but one can easily be created if desired. In TSO, both CALL and command processor (CP) invocation are supported. For the non-POSIX environments, the load module or program object name is DBV. In a POSIX shell (including OMVS under TSO), use the `dbv` command (in lower case) to invoke this utility.

All input to the utility is via command line parameters or the PARM field. A SYSIN DD statement is required, but it can be coded as DUMMY. The FILE= parameter can specify a DD name, as shown in the following example, or a data set name.

```
//*---------------------------------------------------------------*
//*                                                               *
//*                ORACLE DBVERIFY BATCH PROCESSOR                *
//*                                                               *
//*---------------------------------------------------------------*
//*
//ORADBV   EXEC PGM=DBV,
//   PARM='/DD/DBFILE START=1 END=50'
//STEPLIB  DD  DISP=SHR,DSN=ORACLE.V10G.CMDLOAD
//ORA$LIB  DD  DISP=SHR,DSN=ORACLE.V10G.MESG
//DBFILE   DD  DISP=SHR,DSN=ORACLE.ORA1.SYSTEM.BKUP.DBF1
//SYSIN    DD DUMMY
```

On z/OS, DBV produces a zero return code if processing was successful and if no logical or physical errors were detected. Otherwise, return code 8 is produced.

8 Oracle Net

Oracle Net for z/OS supports network communications between Oracle applications and Oracle database systems across different z/OS systems and different operating systems. Oracle provides two listeners on z/OS, an OSDI listener (ORANET) and a generic listener (TNSLSNR). This chapter describes the two listeners and how to configure them. For more information on Oracle Net, refer to the *Oracle Net Services Book Set*.

The following topics are included:

Overview

Oracle provides two listeners on z/OS, an OSDI listener and a generic listener. On z/OS, Oracle Net is implemented as a z/OS OSDI service running in its own address space separate from the Oracle service. The OSDI service acts as a listener for the Oracle instances. All protocol-specific code runs inside the OSDI listener.

The OSDI listener (ORANET), also referred to as the Net service, runs as a service under an OSDI subsystem. Previously, TCP and LU62 connections by Oracle applications, both client and server, were performed through the Net or Net8 service. Starting with Oracle9*i* release 2, all Oracle clients on z/OS open their own sockets and do not require a Net Service, as in prior releases.

The OSDI listener’s primary function is to listen for inbound remote connections to an Oracle instance. For compatibility purposes, the OSDI listener still provides outbound connectivity services for Oracle9*i*, R1 and Oracle8*i*, 8.1.7 Oracle clients.

The generic listener is the Oracle listener (TNSLSNR) that runs in a UNIX System Services shell environment. It provides additional functionality that is not present in the OSDI listener. In particular, it provides support for external routines and shared servers.

Configuring the OSDI Listener

To create a listener under OSDI, you must first define the OSDI listener as a service using the OSDI DEFINE SERVICE command. In addition to defining the service, two other items that must be set up before the service can be started are: a JCL procedure, and network protocol-specific (TCP/IP) configuration. After you have defined OSDI listener as a service and have set up the additional items, you can start the service, which creates z/OS address spaces based on controls that you have specified.

Network Service Definition

The OSDI DEFINE SERVICE command is described completely in Appendix A, "OSDI Subsystem Command Reference". Here, we describe DEFINE parameter considerations that are specific to the OSDI listener.

Service Name

The service name for OSDI listener can be anything that you want within the content limitations described in Appendix A.

TYPE

The TYPE parameter for a database service must be specified as Net.

PROC

This procedure specifies the name of a service JCL procedure that you will place in one of your system procedure libraries. The procedure need not exist when

DEFINE SERVICE is issued, but it must be in place before the service is started. The procedure name can be anything that you choose or that the naming standards of your installation require. The requirements for this procedure are discussed in section "OSDI Listener Region JCL" on page 8-4.

PARM

The PARM string is used to specify additional initialization parameters that are specific to the OSDI listener. These parameters are in the form of keywords and determine which protocols are initialized at OSDI listener startup as well as configuration and debugging features.

The following table lists and describes the OSDI listener keywords:

Table 8–1 OSDI Listener Keywords

Keyword	Description
HPNS	Specifies support for the TCP/IP protocol.
ENCLAVE(SESS|CALL)	Specifies the duration of the enclave. When SESS is specified the enclave is created at logon and deleted at logoff. When CALL is specified the enclave is created when the server is sent a request, and is deleted when the server waits for a receive.
PORT(*nnnn*)	Specifies the TCP/IP port number `(`*nnnn*`)` on which to listen for incoming connections. The default is 1521.
GTF	May be specified at the request of Oracle Support Services. This allows the OSDI listener internal trace to be captured to the z/OS Generalized Trace Facility.
DUMP(*nodename*)	Specifies the high level node, or nodes, of transaction dump data set names. The character string can be up to 26 characters in length, must follow the rules for z/OS data set names, and must not end with a period. When an OSDI listener transaction dump occurs, then the value defined here will be prefixed to a string that includes a time and date stamp to generate a unique data set name. The default is `ORACLE.TRANDMP`.
OSUSER	Specifies that the client operating system USERID and program name should be passed to the OSDI server. This is used by SMF and the logon exit.

Example of OSDI Listener Definition

```
DEFINE SERVICE NET TYPE(NET) PROC(NET) -
DESC('Oracle Network Service') -
SID(NET) -
```

```
PARM('HPNS GTF PORT(1521) DUMP(ORACLE.TRANDMP)')
```

Note: The entire `PARM()` string must be on one line.

OSDI Listener Region JCL

As with a database service, a JCL procedure must be placed in a system procedure library prior to attempting a start of the service. The EXEC card of the JCL must be equivalent to the following:

```
//NET     EXEC PGM=ORANET,REGION=0M
```

`REGION=0M` is specified to ensure that the service can allocate as much private virtual memory as it needs. Some z/OS systems may prohibit or alter a REGION parameter such as this, so you might want to check with your systems programmer to determine if any changes must be made to allow the system to accept your REGION parameter. In addition, the following DD statements are required:

STEPLIB:

This DD statement should be the same as specified for the database service. Refer to "Database Region JCL".

NET8LOG:

Connection-related informational messages, warning messages, and error messages are written to this sequential output file. Oracle Corporation recommends that it also be assigned to a JES spool file.

Note: If the IBM TCP/IP protocol is used, the OSDI listener JCL procedure name must have an associated z/OS userid. Refer to the next topic, "TCP/IP Network Considerations", for details.

Example of OSDI Listener Procedure JCL

```
//NET EXEC PGM=ORANET,REGION=0M
//STEPLIB DD DSN= ORACLE.V10G.AUTHLOAD,DISP=SHR
//NET8LOG DD SYSOUT=X
```

Example of NET8LOG output

```
2000034 09:50:35.0 MIN0017I message service subtask initialized
```

```
2000034 09:50:35.0 MIN0016I command service subtask initialized
2000034 09:50:35.1 MIN0018I bind/unbind service subtask initialized
2000034 09:50:35.2 MIN0026I timer service subtask initialized
2000034 09:50:35.2 MIN0002I networking service NETC     initialization complete
2000034 09:50:35.2 MIN0005I global vector is at 19F0A000
2000034 09:50:35.2 MIN0024I connected to WLM subsystem OSDI
2000034 09:50:50.4 MIN0700I HPNS INITAPI call performed.  RC=0000, EC=00000
2000034 09:50:50.5 MIN0724I HPNS GHBY INITAPI call performed.  RC=0000, EC=00000
2000034 09:50:51.1 MIN0728I HPNS KID INITAPI call performed.  RC=0000, EC=00000
2000034 09:50:51.1 MIN0728I HPNS KID INITAPI call performed.  RC=0000, EC=00000
2000034 09:50:51.1 MIN0728I HPNS KID INITAPI call performed.  RC=0000, EC=00000
2000034 09:50:51.2 MIN0728I HPNS KID INITAPI call performed.  RC=0000, EC=00000
2000034 09:50:51.2 MIN0728I HPNS KID INITAPI call performed.  RC=0000, EC=00000
2000034 09:50:51.2 MIN0728I HPNS KID INITAPI call performed.  RC=0000, EC=00000
2000034 09:50:51.2 MIN0713I I am listening on port 01522 socket 00000
2000034 10:05:58.8 MIN0733I Socket 0000 connected Subtask Kid1, IP 144.025.040.217, Port 01129.
2000034 10:05:58.8 MIN0733I Socket 0000 connected Subtask Kid2, IP 144.025.040.217, Port 01130.
2000034 12:00:13.9 MIN0098I networking service NETC     termination in progress
2000034 12:00:18.9 MIN0722I HPNS Kid #003 shut down.
2000034 12:00:18.9 MIN0722I HPNS Kid #001 shut down.
2000034 12:00:18.9 MIN0722I HPNS Kid #006 shut down.
2000034 12:00:18.9 MIN0722I HPNS Kid #002 shut down.
2000034 12:00:18.9 MIN0722I HPNS Kid #005 shut down.
2000034 12:00:18.9 MIN0722I HPNS Kid #004 shut down.
2000034 12:00:18.9 MIN0723I HPNS Gethostbyname subtask ended.
2000034 12:00:18.9 MIN0721I HPNS shut down, GoodBye.
2000034 12:00:18.9 MIN0091I timer service subtask terminated
2000034 12:00:18.9 MIN0095I bind/unbind service subtask terminated
2000034 12:00:18.9 MIN0093I command service subtask terminated
2000034 12:00:18.9 MIN0094I message service subtask terminated
MIN0000I End of Net8 Log.
```

TCP/IP Network Considerations

The MAXFILEPROC and MAXSOCKETS parameters (under AF_INET) in the BPXPRM*xx* member of SYS1.PARMLIB must be set high enough to support the expected connection load. Both of these parameters can limit the number of connections that the OSDI listener will be able to open. Also, the OSDI listener JCL procedure name must have an associated z/OS userid in order to use TCP/IP, which is controlled by UNIX System Services. The userid must have an OMVS RACF segment (or equivalent, if a product other than RACF is used) if the installation is not using a default OMVS segment.

In addition, the interface resolves names through the standard GETHOSTBYNAME API. Thus the resolution depends on how IBM TCP/IP is configured. If a DNS is defined to TCP/IP, then it will be used. Otherwise, TCP/IP will default the processing to its SITEINFO file. Also, IBM's Language Environment runtime library (LE) must be available through a STEPLIB DD or linklist to the OSDI listener address space in order for `GETHOSTBYNAME` to work. This is an IBM requirement. TNS does a `GETHOSTBYNAME` call at startup to test the function. This call may take minutes to complete if a busy name server is involved. The interface is not ready for work until the MIN0713I message is displayed on the system console. For more information on the GETHOSTBYNAME API, refer to the relevant IBM documentation on TCP/IP.

Operating the OSDI Listener

The OSDI listener is started by the OSDI subsystem start command, for example:

```
ORSS START NET
```

This command would start the OSDI listener defined in the earlier example for "Example of OSDI Listener Definition" on page 8-3 if the subsystem were named 'ORSS'. You should then see the OSDI listener PROC start up followed by the following messages from the OSDI listener address space:

```
MIN0001I networking service initializing
MIN0002I networking service NET  initialization complete
MIN0713I I am listening on port 01521 socket 00000
```

Additional messages are written to the NET8LOG DD, but message traffic to the console is limited to error and warning messages.

Commands for communicating with a running Net service are issued using the z/OS MODIFY (or F) system operator command with the following format, where *name* is the jobname or identifier of the OSDI listener, *cccc* is the command verb, and *pppppp* is a parameter for the command verb:

```
F name,cccc pppppp
```

Command verbs and parameters for the z/SO MODIFY (or F) system operator command are listed in the following table:

Table 8–2 Command Verbs for z/OS MODIFY (or F) System Operator Command

Command Verb	Parameter	Description
start	`hpns`	Starts support for the specified protocol in the OSDI listener.
stop	`hpns`	Stops support for the specified protocol.
dis	`tcp | all | pool`	Display information about existing connections for the specified protocol or storage pool statistics.

The OSDI listener can be stopped with the z/OS stop command (STOP or P), as in '`p net`', or via the OSDI subsystem stop command, as in '`ORSS STOP NET`'. In either case, the following messages will be seen on the console, assuming both protocols were active:

```
MIN0098I networking service NET termination in progress
MIN0721I HPNS shut down, GoodBye.
MIN0099I networking service termination complete
```

The OSDI listener will also respond to the OSDI subsystem '`display`' and '`display long`' commands with appropriate information from the address space. Finally, the OSDI subsystem '`drain`' command will prevent any new connections on either protocol. Existing connections will not be affected. The OSDI subsystem '`RESUME`' command will restore the ability of clients to establish new connections through the OSDI listener.

Formatting OSDI Listener Trace Files

The OSDI listener provides a utility program called TRCASST that formats the trace files the OSDI listener can produce. You may be asked to run TRCASST to help gather diagnostic information required by Oracle Support Services. Sample JCL for TRCASST is provided in *`oracle_hlq`*.SRCLIB(TRCASST).

Before you use TRCASST, ensure that the trace files have not been created with carriage control. TRCASST will be unable to process such files.

When TRCASST runs, the TNSUSMSG DDname must point to a PDS containing a TNSUS message file. This file was placed into *`oracle_hlq`*.MESG(TNSUS) during OSDI listener installation.

Oracle Advanced Security Option Encryption

The OSDI listener supports CHECKSUM and encryption algorithms. The following sections describe a basic method of verifying this feature, if it is to be used by your site. The easiest way to tell if Oracle Advanced Security Option (ASO) encryption is attempting to work is to deliberately set wrong configuration parameters and attempt a connection between the server and client. Incorrect parameters cause the connection to fail.

After receiving the expected failure message, set the configuration parameters to the correct settings and try the connection again. ASO encryption is working properly if no further error messages are received.

The following procedures test ASO encryption by this method. The incorrect parameter settings produce error 12660.

Setting Up ASO Encryption for Test

This section contains information about setting up ASO encryption for testing.

Checklist for Setting Up ASO Encryption

1. Set ASO encryption parameters for the server
2. Set ASO encryption parameters for the client

Step 1: Set ASO Encryption Parameters for the Server

Use ISPF to edit the OSDI listener configuration file on the z/OS system (server system) to add the following parameters and values. If the server is remote (not z/OS), then use the appropriate editor for the server platform to change the `sqlnet.ora` file.

```
SQLNET.CRYPTO_CHECKSUM_SERVER = REJECTED
SQLNET.ENCRYPTION_SERVER = REJECTED
SQLNET.CRYPTO_CHECKSUM_TYPES_SERVER = (MD5)
SQLNET.ENCRYPTION_TYPES_SERVER = (DES40,RC4_40)
SQLNET.CRYPTO_SEED = "abcdefg"
```

The value shown for `SQLNET.CRYPTO_SEED` is only an example. Set it to the value you want. Refer to the *Oracle Advanced Security Administrator's Guide* for more information.

Step 2: Set ASO Encryption Parameters for the Client

Edit the OSDI listener configuration file on the client system to add the following parameters:

```
SQLNET.CRYPTO_CHECKSUM_CLIENT = REQUIRED
SQLNET.ENCRYPTION_CLIENT = REQUIRED
SQLNET.CRYPTO_CHECKSUM_TYPES_CLIENT = (MD5)
SLQNET.ENCRYPTION_TYPES_CLIENT = (DES40,RC4_40)
SQLNET.CRYPTO_SEED = "abcdefg"
```

The value shown for `SQLNET.CRYPTO_SEED` is only an example. Set it to the same value used on the server system.

Testing ASO Encryption

After completing Steps 1 and 2 of the configuration procedure, you are ready to test the operation of the ASO encryption.

Checklist for Testing ASO Encryption

1. Connect client and server
2. Reset configuration parameters on server

Step 1: Connect Client and Server

Attempt a connection between the server and client systems. You receive the following error message:

```
ORA-12660: Encryption or crypto-checksumming parameters incompatible
```

Step 2: Reset Configuration Parameters on Server

Change the ASO encryption parameters on the server to:

```
SQLNET.CRYPTO_CHECKSUM_SERVER = REQUIRED
SQLNET.ENCRYPTION_SERVER = REQUIRED
```

Attempt the connection between the client and server again. If no error message is returned and the connection completes, then ASO encryption is working properly.

Generic Listener

The generic listener, TNSLSNR, is required for Secure Socket Layer (SSL) connections to the RDBMS and Oracle external procedures. SSL connections also require Oracle shared server support. TNSLSNR runs in the POSIX shell environment on z/OS. For more information about the generic listener, refer to the *Oracle Net Services Administrator's Guide*.

Note: When running both the `extproc` agent and shared servers, a separate listener should be used for each.

Oracle Shared Servers

On UNIX, the shared server architecture allows a relatively small number of Oracle server processes to support a much larger number of client sessions. For more information about shared server architecture on UNIX, refer to *Oracle Database Concepts*.

Although Oracle for z/OS does not employ a UNIX-like process model, the shared server configuration can be run within an OSDI service address space and is required for SSL connections. Except for the support of SSL connections, the use of shared server architecture on z/OS is not recommended. Shared servers on z/OS have the following limitations and restrictions:

- All work is done using the velocity goal of the service address space.
- All shared servers execute in a single address space.
- Some UGA (User Global Area) is allocated from the SGA, which increases the size of the SGA and reduces the amount of the address space available to non-shared server work.
- All sockets are owned by a single TCB no matter how many dispatchers are configured.
- TCP connections are managed with the `poll()` socket call instead of asynchronous I/O .
- Shared server sessions may deadlock if all available servers become blocked on a lock.

Oracle External Procedures

Oracle externalprocedures are functions or procedures written in a third-generation language that can be called from PL/SQL code. The supported languages are C and Java. The user-written applications must be invoked in a POSIX shell environment and must be in DLL (dynamic loadable library) form. The use of external procedures requires the generic listener to listen for external procedure calls. For information on configuring the generic listener for external procedures, refer to "Configuring the Generic Listener for External Procedures" on page 8-15. For more information about external procedures, refer to the *Oracle Database Application Developer's Guide - Fundamentals*.

Configuring the Generic Listener for SSL

To set up an Oracle Net SSL connection, you must use shared servers and the Oracle generic listener, TNSLSNR, running in a POSIX shell environment. The following steps describe the files that must be configured for SSL connections.

The generic listener configuration steps are as follows:

- Step 1: Edit the Server Init.ora File on page 8-11
- Step 2: Edit the Server Sqlnet.ora File on page 8-11
- Step 3: Edit the Server and Client Tsnames.ora File on page 8-12
- Step 4: Edit the Listener.ora File on page 8-13

Step 1: Edit the Server Init.ora File

In the server `init.ora` file, specify a dispatcher to support the TCPS protocol. For example:

```
DISPATCHERS="(PROTOCOL=TCPS)"
MAX_DISPATCHERS=1
MAX_SHARED_SERVERS=10
LOCAL_LISTENER=listener
```

Variables for the previous example are defined, as follows:

`listener` is the name of the listener as defined in the server `tnsnames.ora` file.

Step 2: Edit the Server Sqlnet.ora File

In the server `sqlnet.ora file`, specify the directory location of your wallet. For example:

```
WALLET_LOCATION=
(SOURCE=
(METHOD=file)
(METHOD_DATA=
(DIRECTORY=/u/cwang/wallet/v10)
)
)
```

Step 3: Edit the Server and Client Tsnames.ora File

In the server and client `tnsnames.ora` files, specify the connect descriptor to the listener.

In the server `tnsnames.ora` file, you need to add an entry for the listener that was specified in the DISPATCHERS parameters in the server `init.ora` file. For example:

```
listener =
(DESCRIPTION=
(ADDRESS=(PROTOCOL=TCP) (HOST=hostname)(PORT=nnn1))
)
```

Client `tnsnames.ora` file example:

```
ora6ssl =
(DESCRIPTION =
(ADDRESS_LIST =
(ADDRESS = (PROTOCOL = TCPS)(HOST = hostname)(PORT = nnn2))
)
(CONNECT_DATA =
(SERVICE_NAME = service)
)
)
```

Variables for the previous example are defined, as follows:

Table 8–3 Variable Definitions for Code Example

Variable	Definition
listener	is the name of the listener
hostname	is the name of the z/OS machine where the OSDI RDBMS is running
*nnn*1,*nnn*2	are the TCP port numbers on which the generic listener will listen for TCPS connections
service	is the name of the service

Step 4: Edit the Listener.ora File

In the `$ORACLE_HOME/network/admin/listener.ora` file, used by TNSLSNR, specify the ports and protocols that you will listen on, and specify the directory location of the wallet.

On z/OS, you must specify the following so that TNSLSNR will redirect the connection to the server:

```
DIRECT_HANDOFF_listener=no
```

For example:

```
listener =
(ADDRESS_LIST =
(ADDRESS=(PROTOCOL=TCP) (HOST=hostname) (PORT=nnn1) )  << tnslsnr
(ADDRESS=(PROTOCOL=TCPS) (HOST=hostname) (PORT=nnn2) ) << ports
)
DIRECT_HANDOFF_listener = no << redirect connection
WALLET_LOCATION= << wallet locator
(SOURCE=
(METHOD=file)
(METHOD_DATA=
(DIRECTORY=/u/cwang/wallet/v10)
)
)
```

Variables for the previous example are defined, as follows:

Table 8–4 Variable Definitions for Code Example

Variable	Definition
listener	is the name of the listener
hostname	is the name of the z/OS machine where the OSDI RDBMS is running
*nnn*1,*nnn*2	are the TCP port numbers on which the generic listener will listen for TCPS connections
service	is the name of the service

Starting the Generic Listener for SSL and Shared Servers

Perform the following steps to start the generic listener for SSL and shared servers:

1. Verify that the ORACLE_HOME environment variable is set to the same value that was used when you installed the Oracle Database.
2. From the POSIX shell environment, type `lsnrctl start` and press the ENTER key to run the generic listener control program.

The following sample dialogue is typical of what you should expect when you run the generic listener control program with the default listener. Output is prefixed with the character ">". The ">" character will not appear on your screen.

```
Lsnrctl start
> LSNRCTL for OSDI390: Version 10.1.0.2.0 - Production on dd-Mon-yyyy hh:mm:ss
> (c) Copyright 1991, 2004, Oracle. All rights reserved.
>
> Starting $ORACLE_HOME/bin/tnslsnr: please wait...
> TNSLSNR for OSDI390: Version 10.1.0.2.0 - Production
> System parameter file is $ORACLE_HOME/network/admin/listener.ora
> Log messagss written to $ORACLE_HOME/network/log/listener.log
> Trace inforamtion written to $ORACLE_HOME/network/trace/listener.trc
> Listening on: (DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=MVS05.US.ORACLE.COM)(PORT=3044)))
> Listening on: (DESCRIPTION=(ADDRESS=(PROTOCOL=tcps)(HOST=MVS05.US.ORACLE.COM)(PORT=3007)))
> Connecting to (address=(protocol=tcp)(host=mvs05)(port=3044))
> STATUS of the LISTENER
> ------------------------
> Alias LISTENER
> Version TNSLSNR for OSDI390: Version 10.1.0.2.0 - Production
> Start Date DD-Mon-yyyy hh:mm:ss
> Uptime 0 days 0 hr. 0 min. 1 sec
> Trace Level support
> Security OFF
> SNMP OFF
> Listener Parameter File $ORACLE_HOME/network/admin/listener.ora
> Listener Log File $ORACLE_HOME/network/log/listener.log
> Listener Trace File $ORACLE_HOME/network/trace/listener.trc
> Listening Endpoints Summary...
> (DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=MVS05.US.ORACLE.COM)(PORT=3044)))
> (DESCRIPTION=(ADDRESS=(PROTOCOL=tcps)(HOST=MVS05.US.ORACLE.COM)(PORT=3007)))
> This listener supports no services
> The command completed successfully
```

The listener should now be running.

Configuring the Generic Listener for External Procedures

The following steps describe the files that must be configured for external routines in a POSIX shell environment.

As with other components that run in a POSIX shell environment, you will need to set your environment variables correctly, including $PATH, $LIBPATH, and optionally, $TNS_ADMIN. For more information, refer to the *Oracle Database User's Guide*.

On most UNIX systems the RDBMS connection to the `extproc` agent is made through the Oracle IPC protocol (UNIX Domain Sockets). Oracle for z/OS does not support the IPC protocol, so the TCP protocol must be used instead. Since the use of the TCP protocol allows the external procedures to be invoked from anywhere on the network, special care is necessary when configuring the `extproc` agent. It is recommended that the listener used for external procedures be configured as described in the *Oracle Net Services Administrator's Guide*.

The generic listener configuration steps are as follows:

- Step 1: Create or Modify the Tsnames.ora File on page 8-16
- Step 2: Create or Modify the Listener.ora File on page 8-11

Step 1: Create or Modify the Tsnames.ora File

In the server `init.ora` file, specify a dispatcher to support the TCPS protocol. For example, add to the Oracle Server `tnsnames.ora` file an entry similar to the following:

```
EXTPROC_CONNECTION_DATA =
(DESCRIPTION=
(ADDRESS=(PROTOCOL=TCP)(HOST=hostname)(PORT=nnnn))
(CONNECT_DATA=(SID=EXTPROC))
)
```

Variables for the previous example are defined, as follows:

Table 8–5 Variable Definitions for Code Example

Variable	Definition
hostname	is the name of the z/OS machine where the Oracle RDBMS is running
nnn	is the TCP port number on which the generic listener will listen for TCPS connections

Step 2: Create or Modify the Listener.ora File

Add to the `$ORACLE_HOME/network/admin/listener.ora` file, used by TNSLSNR, an entry similar to the following:

```
listener =
(address_list =
(address=(protocol=tcp) (host=hostname) (port=nnnn) )
)
sid_list_listener =
(sid_list =
(sid_desc =
(sid_name = extproc)
(oracle_home = oraclehome)
(program = extproc)
)
)
```

Variables for the previous example are defined, as follows:

Table 8–6 Variable Definitions for Code Example

Variable	Definition
listener	is the name of the listener
hostname	is the name of the z/OS machine on which the Oracle RDBMS is running
nnnn	is the TCP port number on which the generic listener will listen for TCPS connections
oraclehome	is the value of the ORACLE_HOME environment variable to use for the `extproc` agent.

Starting the Generic Listener for External Procedures

Perform the following steps to start the generic listener for external procedures:

1. Login to a POSIX shell environment with the userid under which external procedures should run.
2. Verify that the ORACLE_HOME environment variable is set to the same value that was used when you installed the Oracle Database.
3. From the POSIX shell environment, type `lsnrctl start listener` and press the ENTER key to run the generic listener control program.

The following sample dialogue is typical of what you should expect when you run the generic listener control program with the default listener. Output is prefixed with the character `">"`. The `">"` character will not appear on your screen.

```
Lsnrctl start
> LSNRCTL for OSDI390: Version 10.1.0.2.0 - Production on dd-Mon-yyyy hh:mm:ss
> (c) Copyright 1991, 2004, Oracle. All rights reserved.
>
> Starting $ORACLE_HOME/bin/tnslsnr: please wait...
> TNSLSNR for OSDI390: Version 10.1.0.2.0 - Production
> System parameter file is $ORACLE_HOME/network/admin/listener.ora
> Log messagss written to $ORACLE_HOME/network/log/listener.log
> Trace inforamtion written to $ORACLE_HOME/network/trace/listener.trc
> Listening on: (DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=MVS05.US.ORAC
Lsnrctl start extproc_lsnr
> LSNRCTL for OSDI390: Version 10.1.0.2.0 - Production on dd-Mon-yyyy hh:mm:ss
> (c) Copyright 1991, 2004, Oracle. All rights reserved.
>
> Starting $ORACLE_HOME/bin/tnslsnr: please wait...
> TNSLSNR for OSDI390: Version 10.1.0.2.0 - Production
> System parameter file is $ORACLE_HOME/network/admin/listener.ora
> Log messagss written to $ORACLE_HOME/network/log/listener.log
> Trace inforamtion written to $ORACLE_HOME/network/trace/listener.trc
> Listening on: (DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=MVS05.US.ORACLE.COM)(PORT=3044)))
> Connecting to (address=(protocol=tcp)(host=mvs05)(port=3044))
> STATUS of the LISTENER
> ------------------------
> Alias extproc_lsnr
> Version TNSLSNR for OSDI390: Version 10.1.0.2.0 - Production
> Start Date DD-Mon-yyyy hh:mm:ss
> Uptime 0 days 0 hr. 0 min. 1 sec
> Trace Level support
> Security OFF
> SNMP OFF
> Listener Parameter File $ORACLE_HOME/network/admin/listener.ora
> Listener Log File $ORACLE_HOME/network/log/listener.log
> Listener Trace File $ORACLE_HOME/network/trace/listener.trc
> Listening Endpoints Summary...
> (DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=hostname)(PORT=nnnn)))
> Services Summary
> Service "extproc" has 1 instance(s).
>   Instance "extproc", status UNKNOWN, has 1 handler(s) for this service.
> The command completed successfully
```

The listener should now be running.

9 Security Considerations

This chapter describes postinstallation and operational z/OS security issues in Oracle Database for z/OS, Oracle Net, and associated products and features on z/OS. The information is presented with the assumption that IBM z/OS Security Server (RACF) is being used, but any z/OS product that fully implements IBM's System Authorization Facility (SAF) can be used. If your installation does not use RACF, please refer to your security product's documentation for matters presented here in RACF terms.

The following topics are included:

Overview

Oracle products and OSDI interface in a number of places with native z/OS security features. Some of the resulting interactions are discussed as installation topics in the *Oracle Database Installation Guide for IBM z/OS*. These topics include RACF resource class considerations, Program Properties and APF authorization requirements for Oracle product modules, and requirements for associating z/OS userids with OSDI-defined services.

Controlling Access to OSDI Subsystem Commands

OSDI subsystem command processing includes an authorization check to confirm that the console or the user is allowed to issue the command. You control access to commands by defining resource profiles to the security subsystem and then by granting access (for specific consoles and users) to those resources. If you do not define resource profiles, then the authorization check returns a "resource unknown" indication to OSDI, and OSDI then allows the command to be processed. Thus, the default behavior (in the absence of any profile definitions) is that any command is allowed from any source. This is not chaotic, as it may sound, because access to command-issuing mechanisms themselves (such as consoles) is usually controlled in most z/OS installations. Base your decision to define profiles and to activate the authorization mechanism upon the security standards and procedures at your installation.

The resource profiles that are used to protect commands should be defined in the resource class that you chose when installing Oracle software, as discussed in *Oracle Database Installation Guide for IBM z/OS*. If you elected to accept the default, then this will be the FACILITY class. Otherwise it will be a class name that you chose and configured for your SAF-compliant security software.

The command authorization resource names are of the form:

```
ssn.cmdverb
```

where `ssn` is the OSDI subsystem name, and `cmdverb` is the full-length OSDI command verb. (Verb abbreviations, such as DEF for DEFINE, must not be used in the resource name.)

The level of authorization that must be granted to users or to consoles in order to enable commands depends on the command. The following table lists all of the command verbs and the authorization level required for each:

Command Verbs And Their Required Authorization Levels

Table 9–1 Command Verbs And Their Required Authorization Levels

Command Verb	Authorization Level
DEFINE	Update

Table 9–1 (Cont.) Command Verbs And Their Required Authorization Levels

Command Verb	Authorization Level
ALTER	Control
SHOW	Read
START	Read
DISPLAY	Read
DRAIN	Read
RESUME	Read
STOP	Read

Controlling Access to OSDI Services

OSDI bind processing, which establishes connections between z/OS address spaces, performs an authorization check to confirm that the binding address space (the "client") is allowed to access the target service. The target service can be an Oracle Database for z/OS instance or an Oracle Net network service running on z/OS. The possible client address spaces are as follows:

- Batch, TSO, or POSIX address space running an Oracle tool or utility or a customer-written Oracle application
- CICS TS or IMS TM region running transactions which access an Oracle database
- Local Oracle database instance that is accessing another local or remote Oracle instance through a database link
- Oracle Network Service accessing a local Oracle instance on behalf of remote (inbound) client applications or database links

The bind authorization check applies in all of these cases. In order for the check for a given target service to be meaningful, resource profiles must be defined to a SAF-compliant security server such as RACF. The profile names incorporate the OSDI service name so that access to each service is separately controlled. When profiles are defined, the z/OS userid that is associated with the client address space must have READ authorization on the target service's profile in order for the bind to be allowed. Two profiles are defined for each service: one for normal application binds (used by batch, TSO, andPOSIX Oracle tools or applications) and one for managed binds (used by CICS TS and IMS TM, and by the Oracle server and Oracle Net when operating as **clients** as described above).

If you do not define resource profiles for a service, then all binds from all address spaces are permitted. Oracle Corporation recommends that you define resource profiles for all services so that bind access is controlled via standard z/OS security mechanisms.

The resource profiles that are used to protect binds should be defined in the resource class that you chose when installing Oracle software, as discussed in *Oracle Database Installation Guide for IBM z/OS*. If you elected to accept the default, then this will be the FACILITY class. Otherwise it will be a class name that you chose and configured for your SAF-compliant security software.

- The name structure for the normal application bind resource profile is:

  ```
  ssn.service.UBIND
  ```

  where `ssn` is the OSDI subsystem name, `service` is the target service name, and `UBIND` is a constant indicating application binds.

- The name structure for the managed binds used by CICS TS, IMS TM, Oracle database links, and Oracle Net is:

  ```
  ssn.service.ABIND
  ```

  where `ssn` is the OSDI subsystem name, `service` is the target service name, and `ABIND` is a constant indicating managed binds.

> **Note:** In addition to subsystem-level authentication of binds, an OSDI database instance uses SAF to control access to the Oracle database server's SYSOPER and SYSDBA system privileges. This mechanism is discussed in the following section.

Controlling Access to Database SYSDBA and SYSOPER Privileges

SYSOPER and SYSDBA are access privileges that are associated with an Oracle database instance. A database session with these privileges can perform operating functions such as starting up and shutting down the database and can perform DBA activities such as managing database and tablespace definitions. The privileges can be requested by including "AS SYSOPER" or "AS SYSDBA" in a CONNECT statement issued to a tool such as SQL*Plus.

For local clients connecting from batch, TSO, or POSIX address spaces through cross-memory, access to the SYSDBA and SYSOPER privileges is controlled using SAF-defined resources on z/OS. These must be defined in the same resource class that you use for binds, as discussed in the previous section. You chose this class when installing Oracle software, as discussed in the *Oracle Database Installation Guide for IBM z/OS*. If you elected to accept the default during installation, then this will be the FACILITY class. Otherwise, it will be a class name that you chose and configured for your SAF-compliant security software. The form of the resource names is:

```
ssn.service.OPER
ssn.service.DBA
```

where:

Table 9–2 Variable Definitions for Code Example

Variable	**Definition**
`ssn`	is the OSDI subsystem name
`service`	is the database service name
`OPER`	is a suffix to be entered exactly as shown
`DBA`	is a suffix to be entered exactly as shown

After the resources are defined, granting "read" authorization on a resource to a given z/OS userid allows a job or session with that userid to connect to the database with the given privilege. You must grant read authority on the OPER resource to the z/OS userids that will startup and shutdown the database.

If you do not define these resource names for a given database instance, then any userid is allowed to connect with SYSOPER or SYSDBA privileges. Oracle Corporation recommends that you define these resources so that the use of Oracle database system privileges is controlled.

If you want remote clients to have access to SYSDBA and SYSOPER privileges, you must create an Oracle password file using the ORAPWD utility described in Chapter 7, "Oracle Database Administration Utilities". SAF authorization cannot be used for remote clients because the user's OS userid cannot be verified and might not even be compatible with SAF expectations.

A separate file is used to provide password verification for remote users because the database may not be mounted or open when the connection is requested. (This occurs when the remote user connects in order to issue STARTUP, for example.)

To use an Oracle password file to control remote client access to SYSOPER and SYSDBA privileges, follow these steps:

1. Create and initialize a password file as described in the section "Oracle Password Utility (ORAPWD) on z/OS".
2. Add an ORAPASSW DD statement to the database service region JCL procedure. Specify the dsname of the VSAM linear data set created in step 1, and specify DISP=SHR.
3. Shut down the Oracle instance and stop the associated service.
4. Set the REMOTE_LOGIN_PASSWORD_FILE parameter to EXCLUSIVE in the instance's `init.ora` parameter file.
5. Restart the database service (with the updated JCL procedure) and startup the Oracle instance.

Database Service Actions Subject to z/OS Authorization

When a database service runs on z/OS, some of its interactions with the operating system may be subject to authorization checks. These checks are performed by the operating system, not by Oracle software, and generally are based on the z/OS userid associated with the service address space. The *Oracle Database Installation Guide for IBM z/OS* describes the z/OS mechanisms that are used to associate a particular userid with a service address space. This section describes the actions that the Oracle server and the OSDI infrastructure take that might be subject to authorization checks on your system. It is your responsibility to make sure that a z/OS userid is associated with a database service, if necessary, and that it has the correct authorizations for the functions it must perform.

Data Set Creation and Deletion

The Oracle server can invoke the z/OS IDCAMS utility to create and to delete VSAM LDS files. Details regarding when this occurs and how you control it are provided in Chapter 4, "Defining z/OS Data Sets for the Oracle Database". If your server will be creating or deleting files, make sure that the associated z/OS userid has the necessary authorization to do so with the data set name structure that you are using.

Data Set Open

The Oracle server performs an update-type open on the VSAM LDS files comprising the database. It also opens the alert log and other diagnostic logs for output and opens various parameter and SQL files (such as the SQLBSQ and

ORA$FPS DDs) for input. The associated z/OS userid must have the required authorization for these opens.

OSDI Bind Authorization

Database links are Oracle's mechanism for distributed database access. When an Oracle application uses a database link, a connection is made from one Oracle database instance to another. When this happens on z/OS, an OSDI bind is issued from the first instance to the second (if the second instance is on the same z/OS system) or from the first instance to an Oracle Net network service (if the second Oracle instance is remote). If you are using OSDI bind authorization checking as described previously in the section "Controlling Access to OSDI Services", the z/OS userid that is associated with the first instance must be authorized to bind to the second instance or to Oracle Net.

UNIX System Services Access

Certain Oracle Database for z/OS features (such as the UTL_FILE and UTL_HTTP packages, Oracle JVM, and the Oracle external table feature) use UNIX System Services. To use UNIX System Services, the database service must have an associated z/OS userid, and that userid must have a default OMVS segment defined to the security system. If either of these conditions is not met, then message MIR0110W is issued during service address space initialization, and the features which rely on UNIX System Services are inoperative in that Oracle instance.

External Data Mover Actions Subject to z/OS Authorization

When you use Oracle Recovery Manager (RMAN) to perform backup or recovery actions on Oracle Database for z/OS, one or more separate External Data Mover (EDM) address spaces may be started to perform data movement and backup management tasks. The details of this activity are in Chapter 6, "Database Backup and Recovery". Although it is not defined as an OSDI service, the EDM runs as a system address space and can have an associated z/OS userid via the same mechanisms as OSDI services.

The EDM address space must have the necessary authorization to open sequential backup file data sets for output (during backup) or for input (during recovery). Certain RMAN backup maintenance activities cause the EDM to delete backup data sets via an IDCAMS DELETE command, so the EDM that is used during backup maintenance should have that authorization as well.

Note: Be aware that for backup and restore operations, the EDM address space does **not** open or access the associated Oracle database (VSAM LDS) files. Those files are accessed only in the Oracle server address space.

Oracle Net Actions Subject to z/OS Authorization

The Oracle Net network service JCL procedure name must have an associated z/OS userid. The associated z/OS userid must have an OMVS RACF segment (or equivalent, if a product other than RACF is used) if the installation is not using a default OMVS segment.

Client Authorizations

IBM's TCP/IP protocol makes use of z/OS UNIX System Services. A z/OS address space that uses IBM TCP/IP must therefore be a UNIX System Services process or be capable of being dubbed. Dubbing occurs automatically when needed, but it requires the z/OS userid associated with the address space to have a default UNIX System Services segment defined in the security subsystem (for example, RACF). If dubbing fails, the network connection will not be opened and the application probably will not succeed. You must ensure that all z/OS clients that connect to remote Oracle servers can be dubbed. If in doubt, contact your z/OS security administrator.

Authorizing Oracle Logon

When an Oracle userid is defined as IDENTIFIED EXTERNALLY it means the user is authenticated by operating system facilities rather than by Oracle. On z/OS, this mechanism works in one of two ways:

- The user logs on to the operating system and is authenticated by normal operating system security. When the user runs an Oracle tool or application, it logs on to Oracle with "/" (a single forward slash) instead of specifying an Oracle userid and password. Oracle takes the user's z/OS userid as the Oracle userid (possibly with a prefix, specified as the OS_AUTHENT_PREFIX `init.ora` file parameter). This means the user can access Oracle only with z/OS userids for which the password is known. This technique is normally used only when the user is running on the same z/OS system as the Oracle server being accessed. Although it can be enabled for use over Oracle Net, doing so is not considered secure because the server cannot guarantee that the associated userid was authenticated.

- The user runs an Oracle tool or application that logs on to the Oracle server with an explicit userid/password. The verification of the userid and password is controlled by the LOGON_AUTH database region parameter, which can specify one of three verification choices:

  No SAF check. If a user defined to Oracle as IDENTIFIED EXTERNALLY attempts to logon with an explicit userid/password, the logon is rejected.

  Built-in SAF check. This built-in SAF check verifies that the userid and password that are provided on the logon are valid. The user need not be logged on to z/OS with the same userid and in fact may be running on a non-z/OS platform (connecting via Oracle Net). The Oracle userid must be defined to the z/OS security subsystem and the given password must be correct for the logon to succeed. A `RACROUTE VERIFY` function is performed by the OSDI code, but no logon exit is called in this case. SAF interfaces with any security manager that you have installed (IBM's RACF, CA's Top Secret or ACF2, an internally developed security manager, or another third-party security manager). This SAF check is designed to work with virtually any security manager and eliminates the need for an external logon exit, unless you want to modify the normal `RACROUTE VERIFY` of a userid and password.

  External logon exit. This method calls an external, dynamically loaded logon exit. Oracle Corporation supplies a sample logon exit (which does what the built-in SAF check does), or you can write your own. This exit performs the actions that you specify and then returns success or failure of the validation.

> **Note:** The supplied sample logon exit and the built-in SAF check are functionally equivalent. Unless site-specific changes to the sample logon exit are needed, the built-in SAF check should be used because it is more efficient.

The built-in SAF check and the external logon exit are called only for users that are `IDENTIFIED EXTERNALLY`. These checks are not performed for a user who is defined to Oracle (`IDENTIFIED BY` *<password>*), because these users can be resolved internally within Oracle.

The type of validation done for users that are `IDENTIFIED EXTERNALLY` is indicated in the OSDI service parameters. A single parameter controls this validation. The syntax is:

```
LOGON_AUTH (auth)
```

where *auth* is:

NONE - explicit specification of userid/password not allowed

SAF - perform built-in SAF check

exitname - call logon exit

The default (if nothing is specified) is NONE.

For example:

```
LOGON_AUTH(NONE)
LOGON_AUTH(SAF)
LOGON_AUTH(RACFSMPO)
```

The logon exit must reside in the STEPLIB or JOBLIB concatenation or in the linklist, and it must be in an authorized library.

A sample user logon exit is provided in Oracle SRCLIB library member RACFSMPO. It uses the z/OS SAF interface to invoke the z/OS security manager. If the z/OS security manager does not support the SAF interface, then the calls to RACROUTE in the exit must be replaced with the equivalent calls appropriate for the z/OS security manager.

The calling sequence for the logon exit uses standard z/OS assembler calling conventions. R15 is the entry point, R14 is the return address, R13 points to a standard 72-byte save area, and R1 is the address of a parameter list. The parameter list consists of a list of addresses of each parameter (all values are passed by reference, not by value), and the last parameter has the high-order bit set.

When returning, R15 should be set to 0 to indicate a successful verification of the userid and password that were supplied, and should be set to any nonzero value to indicate any type of failure (four would be an appropriate value).

The exit is called in 31-bit addressing mode, supervisor state, storage protection key 7, and in an authorized address space. The exit will be running in TCB mode with no locks held and with no ARRs, FRRs, or EUT-style FRRs set. The exit is called in primary addressing mode with HASN=PASN=SASN (home, not cross memory mode).

The logon exit should be fully reentrant code.

The parameter list that is passed contains pointers to the following parameters, all of which are input only:

Table 9–3 Input-Only Parameters

Field	Type/Length	Description
work area	char/4k	set to all x'00' before every call to the exit
userid	char/1+	userid to be validated (number of bytes varies)
userid length	bin/2	length of userid
password	char/1+	password to be validated (number of bytes varies)
password length	bin/2	length of password
z/OS jobname	char/8	z/OS jobname from JOB card of client
ASID	bin/2	address space id of client address space
OSDI session id	bin/4	a unique OSDI session id
OS username	char/8	the operating system username (batch, TSO, CICS, and IMS only)
terminal name	char/8	terminal name
program name	char/8	program name
RACF group name	char/8	RACF group name
connection type	char/8	connection type (BATCH, TSO, CICS, IMS, TCP/IP, VTAM) to indicate environment of client
JES jobid	char/8	JES job identifier (such as JOB08237)
job card entry time	bin/4	entry (submission) time of job. Binary hundredths of a second since midnight
job card entry date	packed/4	entry (submission) date of job. Packed decimal 0CYYDDDF, where C=0 FOR 19, C=1 FOR 20, YY=year, DDD=day number within the year (Jan 1=1)
job card accounting info	char/145	from jobcard
network data (high bit set in parameter list)	char/2+	variable length NIV data (refer to Chapter 10, "Oracle SMF Data")

The only output of the logon exit is the R15 return code. No other value that is passed in the parameter list should be modified except the first one.

The first parameter is a 4096-byte work area that is set to all `x'00'` before every call to the logon exit. The logon exit can use this storage for anything that it needs. It should not be freed.

The logon exit can do any of the following:

- call a security manager (SAF calls, RACF, Top Secret, ACF2, and so forth)
- get and free storage using the STORAGE macro (the exit must keep track of all acquired storage and must make certain to free it before exiting)
- call SMF to write SMF records
- call WTO to write messages to the console

The logon exit should not do anything that would cause it to wait for any significant period of time (more than one tenth of a second, for example). Avoid opening data sets, writing to the operator with a reply (WTOR), and creating enqueues.

Any resources that are acquired in the logon exit must be freed before it returns. There is no cleanup call made to the logon exit, so any resources that are not released will accumulate in the address space and could eventually cause resource shortages.

10 Oracle SMF Data

The IBM System Management Facility (SMF) provides a facility for users to collect and record a variety of system and job-related information. SMF formats the information into a number of different records. By creating analysis and report routines, installations can use the information in SMF records to track system usage.

The Oracle server uses the standard SMF interface to write user records to the SMF data sets. These user records contain Oracle server accounting and Oracle auditing information allowing Oracle installation sites to charge individual users for the resources they use.

The following topics are included:

Preparing to Record Oracle Accounting SMF Information

Oracle accounting SMF recording is activated by updating the SMFPRM*xx* member of SYS1.PARMLIB using the SYS or SUBSYS option to allow recording of the Oracle accounting user record type. If the SUBSYS option is used, then the SUBSYS name must match the OSDI subsystem name that hosts the database service. Refer to the IBM SMF documentation for information on implementing SMF.

Specifying the Oracle Record Type

The default Oracle user record type is 0 (zero). A zero for this parameter indicates that no SMF accounting statistics record is to be written. You can override the default to any value between 128 and 255 by adding the SMF_STAT_RECNO (abbreviation is SMFSTRCN) to the OSDI database region parameter file. The SMF record number that is chosen must not be the same as the number that is used by any other z/OS software.

Oracle Corporation recommends using SMF record number 204, but any available record number between 128 and 255 may be used.

If this parameter is not specified, or if zero is specified, then no SMF accounting statistics collection or recording is done. This saves some CPU overhead and saves the overhead of the SMF write itself (which is mostly asynchronous work done by the SMF address space, and the in-line overhead is mostly just moving data into SMF buffers).

The OSDI SMF_STAT_RECNO parameter is defined as follows:

SMF_STAT_RECNO | SMFSTRCN

SMF_STAT_RECNO can be added as an OSDI parameter to the OSDI database region parameter data set to override the default record number 0. In the following example, 204 is the new SMF record type:

```
SMF_STAT_RECNO(204)
```

In addition, make sure the OSUSER parameter is specified in the Oracle Net service definition. This ensures that the operating system Userid and Program Name fields are included in the OSDI BIND parameters passed to the database service. For information on the OSUSER parameter keyword, refer to the section "PARM"in Chapter 8, "Oracle Net".

Starting SMF Recording of Oracle Records

SMF recording of Oracle accounting information starts automatically at startup if SMF is activated and if the Oracle record number is specified.

Because the standard system default record types activated for SMF are 128 through 255, and because the recommended number for Oracle accounting (204) is within this range, many sites automatically begin SMF recording of Oracle accounting records when Oracle is installed, and the SMF_STAT_RECNO parameter is supplied.

The service must be stopped and restarted for this parameter to take effect.

Stopping SMF Recording of Oracle Records

The OSDI SMF_STAT_RECNO parameter can be used to stop SMF accounting recording for Oracle. To stop SMF recording for Oracle regardless of what your system tables specify, use:

```
SMF_STAT_RECNO (0)
```

or take the default of 0. The service must be stopped and restarted for this parameter to take effect.

Events that Generate SMF Accounting Records

After SMF accounting recording is turned on, an SMF record is written each time a user logs off (normal termination or SMFINV=SMFNORM), provided SMF was activated when the user logged on.

SMF records are also written on an abend or cancellation of a job if SMFINV=SMFABORT.

If the z/OS system crashes, then SMF records are not written, and the information is lost.

SMF Accounting Recording under CICS TS

When Oracle Access Manager for CICS TS transactions are used to access data on your local Oracle Database for z/OS server, a single thread can be shared by many CICS TS users. When SMF recording is activated, an SMF record is written for a single thread when the thread is dropped. However, SMF recording is not supported when these transactions are used to access data on a remote Oracle server. If a thread is defined with PROTECT set to NO, then the thread is dropped after being idle for a specified time (CINTERVL) in the thread table.

If a thread is defined with PROTECT set to YES, then the thread is dropped when Oracle Access Manager for CICS TS is stopped with the STOP command. Refer to Chapter 11, "Oracle Access Manager for CICS TS", for more information on the STOP command.

For Oracle Access Manager for CICS TS, SMF accounting information is based on the Oracle userid for the CICS TS transaction.

The following is a sample thread definition table:

```
ORACICS TYPE=THREAD,
AUTH=TRANSID,
PROTECT=NO,
TRANSAC=(PGM1,PGM2)
```

If the above thread definition is used, then a sample session is as follows:

```
PGM1
PGM2
PGM1
```

The thread is dropped after being idle for the number of seconds specified in the CINTERVL parameter, and two SMF records are written. One of the records summarizes statistics for all PGM1 transactions, and the other record summarizes statistics for all PGM2 transactions.

Refer to Chapter 11, "Oracle Access Manager for CICS TS" for instructions for configuring thread definition tables in Oracle Access Manager for CICS TS.

Interpreting an Oracle Accounting SMF Record

To interpret an Oracle accounting SMF record, you first need to dump the SMF data set to a sequential data set. You can then write a program that does all of the following:

- Reads the sequential data set
- Selects only records with the Oracle record number
- Accesses the Oracle SMF record fields using the provided DSECT
- Prints the statistics

A sample program named ORAFMTO is provided in the SRCLIB library that you can customize for your installation. Refer to "Sample Formatting Program for SMF Accounting Records" in Chapter 10, "Oracle SMF Data".

The Assembler copy file, ORASMFO, contains DSECTS that map and document the Oracle SMF record fields. The ORASMFO data set member resides in the Oracle SRCLIB library.

The ORASMFO file is divided into these sections:

- Standard record header section, which contains offsets and lengths of the Net and accounting sections
- Correlation section
- OSDI section

- Database engine section
- Net section (if applicable), which contains information about the network origin of clients on an Oracle Net for z/OS TCP/IP protocol network (IBM or SNS/TCPaccess)
- z/OS accounting section (if applicable)

Not all sections are present in all SMF records. For example, the z/OS accounting section is present only in SMF records for batch and TSO users. When a section is present, the SMF record header contains the correct length for that section, which might be release dependent. The length field for non-existent sections contains 0 (zero).

Contents of the SMF Header Section

Table 10–1 contains brief descriptions for the labels in the SMF header section. For a complete layout of the contents of the SMF header section, refer to the DSECT.

Table 10–1 Contents of the SMF Header Section

ORASMF0 Label	Description
SMFHDR	Standard SMF header
SMFHLEN	Total length of SMF record
SMFHSEG	Segment descriptor = 0
SMFHSIN	SYS IND = X’80’ Subsystem info to follow SYS IND = x’40’ Subtype format record
SMFHREC	Record type recommended = 204 (decimal)
SMFHTIM	Timestamp, time binary (0.01 seconds since midnight)
SMFHDAT	Timestamp, date (0cyyydddf) c=0 for 19xx. c=1 for 20xx
SMFHSYS	System id
SMFHSSI	OSDI Subsystem id
SMFHSUB	Record subtype; 1 = accounting record
SMFSRVC	OSDI service name
SMFSESID	OSDI session id
SMFHRSV1	Reserved
SMFNETO	Offset to Net section

Table 10–1 (Cont.) Contents of the SMF Header Section

ORASMF0 Label	Description
SMFACTO	Offset to z/OS accounting section
SMFHRV2	Reserved
SMFNETL	Length of Net section
SMFACTL	Length of z/OS accounting section
SMFHRV3	Reserved

Contents of the SMF Correlation Section

Table 10–2 contains brief descriptions for the labels in the SMF correlation section.

Table 10–2 Contents of the SMF Correlation Section

ORASMF0 Label	Description
SMFAUTH	Authorization id = TSO logon id Batch user id on jobcard CICS USERID, TERM-ID,TRANS-ID, PROGRAM-ID, or OPID
SMFCORI	Correlation id = TSO logon id Batch jobname CICS jobname Not valid for Oracle Net
SMFCONN	Connection type (TSO,BATCH,CICS,VTAM,TCP/IP, IMS)
SMFASID	Users address space id (not valid for Oracle Net
SMFOUSR	Oracle logon id
SMFTNAME	Originating terminal id (if available)
SMFPNAME	Originating program name (if available)
SMFGRPN	RACF group name (if available)
SMFJBID	JES job identifier

Table 10–2 (Cont.) Contents of the SMF Correlation Section

ORASMF0 Label	Description
SMFENTRY	RDR jobcard entry time (batch and TSO only). This field is equivalent to the SMF5RST field in the SMF job termination (type 5) record.
SMFEDATE	RDR jobcard entry date (batch and TSO only). This field is equivalent to the SMF5RSD field in the SMF job termination (type 5) record.

Contents of the SMF OSDI Data Section

Table 10–3 contains brief descriptions for the labels in the SMF OSDI data section.

Table 10–3 Contents of the SMF OSDI Data Section

ORASMF0 Label	Description
SMFTIM	Beginning timestamp, time binary (0.01 second since midnight)
SMFDAT	Beginning timestamp, date (0cyyydddf), ending time and date in header c=0 for 19xx. c=1 for 20xx
SMFDTAI	Data in
SMFDTAO	Data out
SMFXMCPU	Cross memory CPU time (TOD format)
SMFRPCS	RPC count
SMFHWST	High-water mark of storage used
SMFINV	Reason for invocation
SMFNORM	Normal termination
SMFABORT	Clean up done

Contents of the SMF Database Engine Data Section

Table 10–4 contains brief descriptions for the labels in the SMF database engine section.

Table 10–4 Contents of the SMF Database Engine Section

ORASMF0 Label	Description
SMFLRC	Logical read count
SMFPRC	Physical read count
SMFLWC	Logical writes
SMFDMC	DML COMMITs
SMFDMR	DML ROLLBACKs
SMFDED	DEADLOCKs
SMFHDLN	Length of SMF header

Contents of the SMF Net Data Section

Table 10–5 contains brief descriptions for the labels in the SMF Net data section.

Table 10–5 Contents of the SMF Oracle Net Data Section

ORASMF0 Label	Description
SMFNET	Net section header
SMFNETL	Length of Net NIV information. The information contained in this section is specific to the Net driver in use. This information is variable length.
SMFNETA	Start of variable length information. Refer to next section, "Oracle Net Network Information Vector Overview".

Oracle Net Network Information Vector Overview

Oracle Net constructs a Network Information Vector (NIV) list containing information about the network origin of an incoming client connection. This information is available to the Logon User Exit and is also written out in the SMF record for each user.

The vector list is preceded by a 2-byte length field indicating the length of the entire list including the length field itself. The individual vectors in the list consist of a 1-byte length field indicating the length of the vector, a 1-byte vector ID field identifying the vector, and a variable number of vector-specific data bytes. The first NIV in the list is always a protocol identification NIV, which will identify the Net protocol being used as well as the network location of the client.

Currently, only the protocol identification NIV is built. Other NIVs may be added in the future as required. The following tables describe the NIV list and the individual NIV formats.

General NIV Format

Table 10–6 lists and describes the general NIV format:

Table 10–6 General NIV Format

Byte	Contents
0	Total length of NIV including byte 0
1	ID of NIV
2 through p	NIV data

NIV List Format

Table 10–7 lists and describes the NIV list format:

Table 10–7 NIV List Format

Byte	Contents
0 through 1	Total length of NIV list including bytes 0 and 1
2 through m	First NIV
m+1 through n	Second NIV
n+1 through ...	......
.. .-x	......
x+1 through y	LSast NIV

Net TCP/IP Identification NIV Format

Table 10–8 lists and describes the Net TCP/IP identification NIV format:

Table 10–8 Net TCP/IP Identification NIV Format

Byte	Contents
0	NIV length = x’08’
1	NIV ID = x’03’
2 through 3	TCP port number from which client originated
4 through 7	Internet address on which client resides (hex format)

Contents of the SMF z/OS Accounting Data Section

Table 10–9 lists and describes the labels in the SMF z/OS accounting data section:

Table 10–9 Contents of the SMF z/OS Accounting Data Section

ORASMF0 Label	Description
SMFACT	z/OS accounting section header
SMACTNF	Number of accounting fields. This field is equivalent to the SMF5ACTF field in the SMF job termination (type 5) record
SMFACTA	z/OS accounting information. This field is equivalent to the SMF5JSAF field in the SMF job termination (type 5) record
SMFACLN	Length of z/OS accounting section

Sample Formatting Program for SMF Accounting Records

A sample program, ORAFMTO, is provided with the Oracle SMF interface to format Oracle accounting SMF records. ORAFMTO is an Assembler program that reads and formats SMF accounting records with the default Oracle type of 204. It reads records from a variable-blocked sequential data set and writes the formatted records to a fixed-block sequential data set with a logical record length of 133.

The sample ORAFMTO program is in the Oracle SRCLIB library. The following members are included:

Table 10–10 Members in Sample ORAFMTO Program

Member	Description
ORAFMTCL	contains sample JCL to assemble and link ORAFMTO.

Table 10–10 (Cont.) Members in Sample ORAFMTO Program

Member	Description
ORAFMTGO	contains sample JCL to run ORAFMTO.
ORAFMTO	extracts and prints values from the Oracle SMF records.

Table 10–11 lists and describes the SMF record values:

Table 10–11 SMF Record Values

Value	Description
SSN	OSDI Subsystem name
SERVICE	OSDI Service name
SMFAUTH	Authorization id = TSO logon id Batch user id on jobcard CICS USERID, TERM-ID,TRANS-ID, PROGRAM-ID, or OPID
SMFCONN	Connection type (TSO,BATCH,CICS,VTAM,TCP/IP, IMS)
Oracle ID	Oracle user id. (This field is blank if the connection is not associated with a user id.)
DATE	Start date of Oracle session
TIME	Start time of Oracle session
CPU SECONDS	Total CPU seconds used in the Oracle address space (SMFXMCPU)
LOG READS	Count of logical reads
PHY READS	Count of physical reads
LOG WRITES	Count of logical writes
DMC	Data Manipulation Language Commits
DMR	Data Manipulation Language Rollbacks
DED	Deadlocks
HI STG	High-water mark of main storage used by the session

If any of the values are too large for the precision their column allows, then they are shown as a series of asterisks.

The following is sample output from the ORAFMTO program:

```
SSN  SERVICE  SMFAUTH  SMFCONN  ORACLE ID      DATE   TIME          CPU SECONDS   LOG READS  PHY READS LOG
WRITES DMC DMR DED HI STG
---- -------- -------- -------- ------------- ------ ----------- -------------- ---------- ----------
---------- --- --- --- ------
ORA1 ORA108   MJJONES  BATCH    SYS           00.259 08:06:25.82       1.258847       5396        220
33   0   0   0   955K
ORA1 ORA108   MJJONES  TSO      SCOTT         00.259 08:07:16.43       1.604269       1611       1037
8   0   0   0   673K
ORA1 ORA108   MJJONES  TSO      SYSTEM        00.259 08:10:35.49        .318189        614         17
97   3   1   0   675K
ORA1 ORA108   MJJONES  BATCH    SCOTT         00.259 10:12:33.04        .580421       3006         96
23   0   0   0   772K
```

Auditing Database Use

Oracle Database for z/OS allows system-wide audit records to be written to operating system audit trails. Oracle Database for z/OS uses the System Management Facility (SMF) as its operating system audit trail. This section describes z/OS-specific considerations for defining and using an operating system audit trail. Refer to *Oracle Database Concepts* and *Oracle Database Administrator's Guide* for additional information on auditing.

Preparing To Record Oracle Database for z/OS Audit information

Two steps must be performed before SMF audit recording can take place:

1. Two INITORA parameters must be specified.
2. SMF recording by the Oracle database instance must be enabled.

Until both steps are completed no SMF auditing will occur.

Step 1: Specify INITORA Parameters AUDIT_TRAIL. and AUDIT_FILE_DEST

AUDIT_TRAIL=OS is required to inform Oracle Database for z/OS that operating system auditing is desired. AUDIT_FILE_DEST is required to indicate the desired SMF record type.

When specifying a record type, you must select a user SMF record type that does not conflict with any other user record types. This includes the Oracle Database for z/OS accounting record type described in "Specifying the Oracle Record Type", earlier in this chapter. For example, specifying AUDIT_FILE_DEST=205 causes Oracle Database for z/OS audit records to be written to SMF record type 205.

Step 2: Enable SMF Recording by the Oracle Database for z/OS Instance

There are two ways to accomplish this:

1. You can use the SMFPRM*xx* member of SYS1.PARMLIB; or
2. You can use the z/OS SETSMF command.

   For example, issuing the following SETSMF command would cause subsystem ORA1 to begin writing SMF records to user type 205.

   ```
   SETSMF SUBSYS(ORA1,TYPE(205))
   ```

   Continuing the example, if you decide also to write Oracle Database for z/OS accounting records to SMF and use the recommended SMF record type of 204 for those records, then the following SETSMF command activates SMF recording for both record types.

   ```
   SETSMF SUBSYS(ORA1,TYPE(204,205))
   ```

Interpreting Oracle Database for z/OS Audit Records

To interpret Oracle audit SMF records, you first need to dump the SMF data set to a sequential data set. You can then write a program to read the sequential data set and extract the desired records from it. A sample program to extract and print the audit records is provided in member ORAFMTAO of the installed Oracle Database for z/OS SRCLIB data set. It refers to the copy file ORASMFAO, containing DSECTS that map and document the Oracle audit SMF record fields. The contents of the Oracle audit data are defined by Oracle.

You can customize ORAFMTAO for use in your installation. If you select an SMF record type other than 205, then update the value of the ORAREC constant to match your chosen SMF record type and reassemble the program. SRCLIB member ORAFTACL contains sample JCL to assemble and link ORAFMTAO. Sample JCL to run ORAFMTAO is provided in SRCLIB member ORAFTAGO.

11 Oracle Access Manager for CICS TS

This chapter discusses how to configure and operate the Oracle Access Manager for CICS TS. The following topics are included:

Overview

Oracle Access Manager for CICS TS provides the interface between Oracle Precompiler programs (COBOL, C, or PL/I languages) executing Oracle SQL in a CICS TS transaction, and an Oracle database. Oracle Access Manager for CICS TS communicates with an OSDI local database server (but not an MPM database server), or it communicates with a remote Oracle database (but not an MPM database). The target service or remote database is defined in a TNSNAMES entry that is used as input when the thread table is created. Each instance of Oracle Access Manager for CICS TS communicates directly with one OSDI service or one remote database.

All OSDI clients connect to a service address space using the bind mechanism. Oracle Access Manager for CICS TS connects to a service address space at startup time using an OSDI special purpose bind that is specific to multi-connection environments. This provides efficiencies when subsequent connections are created to service any requests from Oracle transactions in the CICS TS region.

Connections for Oracle CICS TS transactions, whether to a local or remote database, are created and reused based on criteria defined in the thread table, and they are assigned to a CICS TS transaction the first time that it accesses Oracle.

When the CICS TS transaction request requires work to be done in the Oracle database, processing is switched to a CICS TS subtask which contains an IBM Language Environment (LE) environment prepared for executing the Oracle client code (LIBCLNTS). The Oracle client code communicates with a local Oracle server via cross-memory services or with a remote server via Oracle networking socket calls to satisfy the request. Starting with Oracle9*i* release 2, this direct use of socket calls eliminates the requirement of a Net Service in earlier releases.

Oracle Access Manager for CICS TS Applications

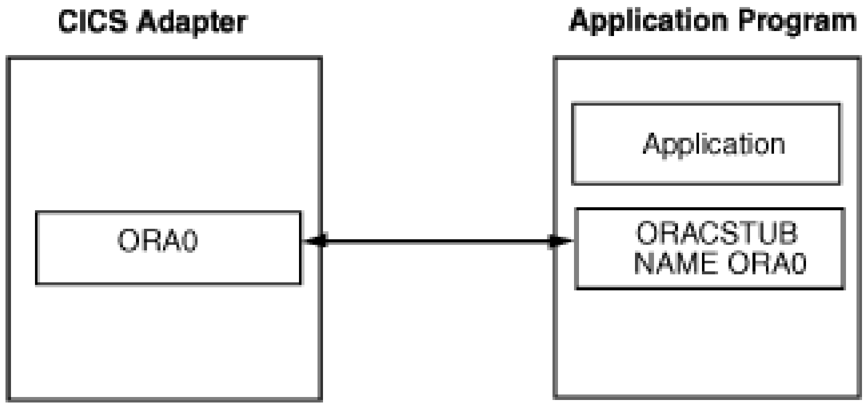
You can run applications built with these Oracle products for z/OS under Oracle Access Manager for CICS TS:

- Pro*COBOL
- Pro*C
- Pro*PL/I

Oracle Access Manager for CICS TS Configuration

The following figure shows the relationship between an application program and Oracle Access Manager for CICS TS.

Figure 11–1 CICS TS Adapter for an Application Program



Application programs are mapped to a particular instance of Oracle Access Manager for CICS TS by way of an adapter name. The adapter name is part of the Oracle stub linked with the application program, and will correspondto the adapter name provided at start up of Oracle Access Manager for CICS TS.

In the figure, ORA0 is the example adapter name. It is assigned on the CICS TS side when Oracle Access Manager for CICS TS is started. Refer to "Step 8: Start Oracle Access Manager for CICS TS Adapter" on page 11-18 and the START command details on page 11-34 for more information on the adapter name. An adapter name gets associated with an application program when it is linked with an ORACSTUB stub. Refer to "Step 6: Generate the ORACSTUB Stub for CICS TS" on page 11-16 for details on generating an ORACSTUB stub.

Configuration

Checklist

Oracle Access Manager for CICS TS must be installed successfully before you can configure it. Refer to the *Oracle Database Installation Guide for IBM z/OS*, for more information. For information on CICS TS, refer to the CICS TS Transaction Server for z/OS documentation for your release.

Configuration Steps

- ❏ Step 1: Define and Assemble Thread Definition Table
- ❏ Step 2: Define the MESG Library to CICS TS

Configuration Steps

Steps to configure the Oracle Database.

Step 1: Define and Assemble Thread Definition Table

A thread represents a connection to the database. It is maintained by Oracle Access Manager for CICS TS. All active CICS TS transactions that communicate with an Oracle server database use a thread.

A table, defined and assembled as a load module, defines threads and their characteristics to Oracle Access Manager for CICS TS. When you create the thread definition table, configure the threads to match the expected workload and priorities for Oracle Access Manager for CICS TS users and applications.

Note: The thread table must be reassembled if it was built on a version prior to release 9.2.

Before defining your table, select a table name appropriate for your installation. The table name is specified to Oracle Access Manager for CICS TS at startup in the Oracle Access Manager for CICS TS START command.

Note: You can also use the thread definition table to create automatic startup and shutdown (PLTPI and PLTSD) programs.

Complete these steps to build the thread definition table.

Step 1.1: Determine Requirements for Each Transaction

Consider these criteria in determining the requirements for each transaction:

- The authorization assignment specified in the AUTH parameter
- The peak and average rates of the transaction
- Whether the transaction requires dedicated threads
- Whether Oracle COMMIT processing or CICS SYNCPOINT is to be used for commit and rollback functions. For more information, refer to "Recovery Considerations" on page 11-23.

Step 1.2: Define Thread Requirements in Thread Definition Table

When you define a thread table:

- Use the TRANSAC parameter to group transactions according to similar characteristics. If a transaction requires dedicated threads, then you should be sure to give it a separate definition.
- Use the COPIES parameter to specify the number of threads you need to define for each group.
- Base the thread definition on the peak and average rates of the transaction. The specified value must be large enough to handle a workload for the number of transactions specified in the TRANSAC parameter.
- Use the PROTECT parameter to specify how to protect the threads defined in the COPIES parameter. The session is not dropped, even if the IDLE TIME has expired.
- Use the WAIT parameter to specify the action to take if no thread is available.

Thread Definition Table Parameters

The following table contains descriptions of the thread definition table parameters:

Table 11–1 Thread Definition Table Parameters

Parameter	Description
AM4COID	is used, if specified, as the Oracle user id for the control thread. It must be a constant character string of 30 characters or less, and requires the AM4CAUTH parameter. If omitted, then the CICS applid is used for autologon. This parameter is specified on the TYPE=START statement and requires that at least one thread be defined using the OID and AUTH parameters.
AM4CAUTH	is the authentication string (password) for the Oracle user id specified in the AM4COID parameter. It must be a constant character string of 30 characters or less. This parameter is specified on the TYPE=START statement.
AUTH	determines how the Oracle user id is derived without an explicit connect statement. The Oracle userid is also used for SMF accounting.
CINTERVL	specifies the control transaction time interval. This interval is used to determine how frequently unprotected idle threads are dropped. Idle threads defined with PROTECT=YES are not affected by the setting of CINTERVL. The interval also affects the length of time it can take Oracle Access Manager for CICS TS to invoke emergency shutdown when the Oracle server cannot be contacted. The value is expressed in minutes and seconds, and valid values range from 0000 to 5959. The default value is 0030 (00 minutes, 30 seconds).
COMMIT	specifies whether to use Oracle COMMIT processing or CICS SYNCPOINT for commit and rollback functions. Valid values are CICS and ORACLE. The default is ORACLE. This parameter can also be specified on the TYPE set to START statement or the Oracle Access Manager for CICS START command. Refer to the section"Recovery Considerations" on page 11-23 for information on these parameter values.
COPIES	indicates the number of threads this definition generates.
DESC	defines the z/OS descriptor code used for the console message. Valid values are numeric. The default is 11. DESC is an optional parameter.

Table 11–1 (Cont.) Thread Definition Table Parameters

Parameter	Description

ENAME	is used in conjunction with TYPE=ENV to specify environment variables for Oracle Access Manager for CICS TS. It is useful for setting the NLS environment variables to values appropriate for your environment. The syntax is:
	ENAME=(var=val, ...)
	where `var` is the name of the environment variable you wish to set, and `val` is the value you wish to give to `var`. You can specify several `var=val` pairs separated by commas in a single ENAME parameter, or a separate TYPE=ENV can be coded for each variable. Do not enclose the names or values in quotes.
	The following example shows a single ENAME parameter being used to specify NLS_LANG and NLS_DATE_FORMAT:
	`ORACICS TYPE=START,SSN=ORA0`
	`ORACICS TYPE=THREAD,COPIES=30,PROTECT=NO,         X`
	`  TRANSAC=*`
	`ORACICS TYPE=ENV,                                  X`
	`  ENAME=(NLS_LANG=AMERICAN_AMERICA.WE8EBCDIC37C,   X`
	`  NLS_DATE_FORMAT=DD-MON-RR)`

FREESPC	defines the amount of free space to be allocated for later additions. This parameter is not implemented.
MAXCRSR	defines the maximum number of cursors used. The MAXCRSR parameter is patterned after the precompiler option MAXOPENCURSORS. Refer to the *Pro*C/C++ Precompiler Programmer's Guide* for more information about the MAXOPENCURSORS option. The default is 10.
MAXTHRD	defines the maximum number of threads allocated for this table definition. If the total threads for the table exceed this value, then the MAXTHRD value is adjusted. The default is 10.
NAME	identifies the adapter you want started. If NAME is not specified, then the started adapter name defaults to the thread table name. This parameter only applies if TYPE is set to PLTPI.
OID	is used, if specified, as the Oracle user id for this thread definition, including all threads generated with the COPIES parameter. It must be a constant character string of 30 characters or less. When specified, the value in the AUTH parameter is used as the associated authentication string.

Table 11–1 (Cont.) Thread Definition Table Parameters

Parameter	Description
`PRIORITY`	specifies whether Oracle Access Manager for CICS TS subtasks are run at a higher or lower priority than the current dispatching priority when subtasking is used. Valid values are HIGH and LOW. If HIGH is specified, then Oracle Access Manager for CICS TS subtasks are run at a higher dispatching priority. If LOW is specified, then Oracle Access Manager for CICS TS subtasks run at a lower dispatching priority. LOW is the default.
`PROGRAM`	specifies the name of the program used with the Oracle Access Manager for CICS TS transaction. ORACICS is the default. This parameter only applies if TYPE is set to PLTSD.
`PROTECT`	specifies whether a thread is protected. Valid values are YES, NO, and `nn`, where `nn` is the number of protected threads when used with the COPIES parameter. A value of YES indicates the thread must be protected. (The session is not dropped, even if IDLE TIME has expired.) A value of NO indicates the thread does not need to be protected. NO is the default.
`ROUTCDE`	defines the z/OS route code to use when writing messages to the z/OS operator console. The default is 11.
`SSN`	specifies the alias name used for the Oracle Net connection (as specified in the TNSNAMES entry used as input to the CIN step of the TBLJCLN job in SRCLIB) for database access. The alias must be four characters or fewer.
`START`	specifies the name of the program for CICS PLTPI processing when TYPE is set to PLTPI.
`STOP`	specifies the name of the adapter stopped during PLTSD shutdown processing. This name corresponds to the value of the NAME parameter used if shutdown processing is performed manually with the STOP command.
`TRANSAC`	specifies the transaction codes eligible to use the thread. A value of * indicates the thread is to be used as a default and no other definitions apply.
`WAIT`	specifies the action to be taken when all threads specified for a transaction are in use. Valid values are YES, NO, and POOL. YES indicates the transaction is placed in a CICS WAIT state until a thread is available. NO indicates a return code is set indicating no threads are available. POOL indicates the general thread pool (TRANSAC=*) is used. YES is the default.

Two additional thread definition table parameters (AUTH and TYPE) are described in the following sections.

AUTH Thread Definition Table Parameter

A maximum of three values can be specified for AUTH. Valid values are listed in the following table:

Table 11–2 Valid Values for AUTH

Value	Description
OPID	based on the operator id.
PROGRAM	causes the Oracle user id to be derived from the program name.
`TERMID`	causes the Oracle user id to be derived from the terminal identification.
`TRANSID`	causes the Oracle user id to be derived from the transaction id.
`USERID`	causes the Oracle user id to be derived from the CICS user id.
authorization string	causes the Oracle user id to be the specified constant character string if the OID parameter is not specified. The implied Oracle user id is derived by prefixing OS_AUTHENT_PREFIX, for example, the string OPS$ to the value specified by this AUTH parameter. If the OID parameter is specified, then this is the authentication string (password) used for that Oracle user id. OS_AUTHENT_PREFIX will not be prefixed to this string. For example, by specifying AUTH='SCOTT', OPS$SCOTT is used as the Oracle user id.

TYPE Thread Definition Table Parameter

TYPE is a parameter specifying the type of entry being defined. Valid values are listed in the following table:

Table 11–3 Valid Values for TYPE

Value	Description
`CONTINUE`	indicates the previous definition is being continued.

Table 11–3 (Cont.) Valid Values for TYPE

Value	Description
ENV	is used in conjunction with the ENAME parameter to specify environment variables for Oracle Access Manager for CICS TS.
END	indicates the end of a table.
START	indicates the beginning of the table and sets default values.
THREAD	indicates a thread is being defined.
PLTPI	indicates PLTPI generation. For PLTPI generation, the only valid value is PLTPI. A value of PLTPI indicates this definition generates the PLTPI program for Oracle Access Manager for CICS TS. Refer to "Step 9.2: Using the PLTPI Program" for an example of a PLTPI generation statement. You must perform PLTPI generation separately from the thread table and PLTSD generation.
PLTSD	indicates PLTSD generation. For PLTSD generation, the only valid value is PLTSD. A value of PLTSD indicates this definition generates the PLTSD program for Oracle Access Manager for CICS TS. Refer to "Step 9.3: Generate the PLTSD Program" for an example of a PLTSD generation statement. You must perform PLTSD generation separately from the thread table and PLTPI generation.

Sample Thread Definition Table

This sample thread definition table resides in the SRCLIB library in the ORACICSD member:

```
ORACICS TYPE=START,                                 X
   SSN=ORA0,                                        X
   MAXTHRD=30,                                      X
   MAXCRSR=5
ORACICS TYPE=THREAD,                                X
   COPIES=4,                                        X
   PROTECT=2,                                       X
   TRANSAC=(TTRN,TRN2,TRN3)
ORACICS TYPE=CONTINUE,                              X
   TRANSAC=(TRN9,TRNA)
ORACICS TYPE=THREAD,                                X
   COPIES=4,                                        X
   PROTECT=NO,                                      X
   TRANSAC=(TRN1,TRNT,TRN4),                        X
   AUTH=(USERID,'STRING')
```

```
ORACICS TYPE=THREAD,                                   X
   COPIES=4,                                           X
   PROTECT=NO,                                         X
   TRANSAC=(TRN5,TRN6,TRN7),                           X
   AUTH='AUTHORIZATION_STRING'
ORACICS TYPE=THREAD,                                   X
   COPIES=4,                                           X
   PROTECT=NO,                                         X
   TRANSAC=TRN8
ORACICS TYPE=THREAD,                                   X
   COPIES=4,                                           X
   PROTECT=NO,                                         X
   TRANSAC=*
ORACICS TYPE=END
   END
```

Specifying User Authorization

You can specify authorization in the thread table, as follows:

- To use an explicit Oracle id for the control thread, use the AM4COID and AM4CAUTH parameters on the TYPE=START. For example, to use control thread userid AM4CICS and authorization ORACLE, include the following parameters:

  ```
  ORACICS TYPE=START,AM4COID='AM4CICS',AM4CAUTH='ORACLE'
  ```

- To define threads to use an explicit Oracle id, use the OID and AUTH parameters on the TYPE=THREAD entry. For example, to use userid SCOTT and authentication TIGER for transactions TTRN, TRN2, and TRN3 in the above example, include the following parameters in the TYPE=THREAD statement:

  ```
  OID='SCOTT',AUTH='TIGER'
  ```

- Use the parameter AUTH=`'authorization_string'` to define threads using a constant character string. The parameter AUTH=`'authorization_string'` derives the Oracle user id and provides the highest level of thread sharing and throughput. AUTH set to TRANSID or AUTH set to PROGRAM performs equally only if one TRANSID or PROGRAM uses a thread definition. This type of authentication is only valid against a local database. For example:

  – `OS_AUTHENT_PREFIX=OPS$` is defined as an `init.ora` file parameter

  - User id `OPS$SCOTT` has been created as `CREATE USER OPS$SCOTT` identified externally
  - To use for transactions TRN9 and TRNA, add the the `AUTH='SCOTT'` parameter in the TYPE=THREAD statement in the previous example.
  - The values in the AUTH parameter are used in the order specified. AM4CICS attempts to reuse a free connected thread. If there are no free threads with an active authorization matching the current request (the same @Oracle user ID), a logon will be performed using the first valid authorization option. Only if the first option is invalid (for example, AUTH=USERID but the user did not signon to CICS), will the second or third authorization options be used.

- An explicit connect statement takes priority regardless of the authentication specified on the TYPE=THREAD statement. However, the thread and database connection is reset at transaction-end with this method and therefore additional overhead is incurred.

Special Considerations

These conditions apply to thread table definitions:

- If a transaction code does not have an associated thread table definition, then the general thread pool definitions (TRANSAC=*) are used.
- If all threads for a transaction are in use and the WAIT parameter specifies POOL, then the thread used has all the characteristics defined in the TRANSAC=* definition. The transaction thread characteristic is not carried over to the pool thread.

Step 1.3: Assemble and Link Thread Table

The sample JCL in SRCLIB library member TBLJCLN assembles and links the threadtable.

Oracle Net allows multiple entries in the TNSNAMES data set. Oracle Access Manager for CICS TS must have its entry as the only entry.

In the INFILE DD statement in the first step (CIN), specify your TNSNAMES data set containing your TNS connect descriptor and alias for CICS. The alias name for CICS must be four characters or fewer and must match the SSN parameter specified in the thread table definition or the SSN override parameter on the START command. There must be only one entry used by CICS. You might need to create a separate PDS member from one of the TNSNAMES entries for the CICS. The service name or alias must be on a separate line from the rest of the connect descriptor. A TNSNAMES specification for Oracle Access Manager might be similar to the following:

ORA0=

```
     (DESCRIPTION=
      (ADDRESS=
       (PROTOCOL=TCP)
       (HOST=HQUNIX)
       (PORT=1533))
       (CONNECT_DATA=(SID=ORA0)))
```

For a local database, the TNSNAMES specification might be similar to the following:

```
ORA0=
     (DESCRIPTION=
      (ADDRESS=
      (PROTOCOL=xm)
      (SID=ORA0)))
```

In the SYSLMOD DD statement, specify the name of the data set where the table load module resides. The load module name is the table name used in the MOD parameter or the START command.

Step 1.4: Installing a Revised Thread Table with CICS TS Executing

If you want to install a revised thread table while CICS TS is executing, then perform these steps:

1. Ensure that you have reassembled the table as described in "Step 1.3: Assemble and Link Thread Table".
2. Issue the STOP command for the adapter.
3. Define the thread definition table to CICS TS using the CEDA command:

   ```
   CEDA DEFINE PROGRAM(tablename) LANG(ASSEMBLER) GROUP(groupname)
   ```

   where:

Table 11–4 Variable Description for Code Example

Variable	Description
tablename	is the name of your thread definition table.
groupname	is the name of the group, typically ORACLE.

1. Issue the CICS TS master terminal command to install the revised thread table where tablename is the name of the revised thread definition table:

   ```
   CEMT SET PRO(tablename) NEW
   ```

2. Issue the START command for the adapter.

Step 2: Define the MESG Library to CICS TS

You can make the MESG Library available to CICS TS in one of two ways:

- Add an ORA$LIB DD statement to the CICS JCL specifying the MESG library data set name.
- Concatenate the MESG library data set name to the CICS STEPLIB DD concatenation.

Note: This library must be authorized to be in CICS STEPLIB.

Step 3: Copy Access Manager for CICS TS Modules to CICS Libraries

- Copy the ORACICSC and LIBCLNTS modules from the Oracle CMDLOAD library to an authorized library in the CICS STEPLIB concatenation. This library must be a PDSE library.
- Copy the ORACICS and CICADPX modules from the Oracle CMDLOAD library to a library in the CICS DFHRPL concatenation or alternatively, add the Oracle CMDLOAD library to the CICS DFHRPL concatenation.
- If you are accessing a remote server, ensure that the CICS address space is dubbed, as describedin the section "Client Authorizations".

Step 4: Define CICS TS to Oracle and Grant Privileges

Oracle Access Manager for CICS TS establishes a database connection at startup

- to perform recovery when CICS syncpoint is used,

- to manage idle connections,
- and to verify that Oracle services are available during execution.

You must define Oracle Access Manager for CICS TS as an Oracle user by one of the following methods:

- If you are using AM4COID and AM4CAUTH in the thread table, then the userid is the value specified in the AM4COID parameter. For example, if AM4COID=AM4CICS and AM4CAUTH=AMTHREAD, then the following statements would be used to define Oracle Access Manager for CICS TS:

  ```
  CREATE USER AM4CICS IDENTIFIED BY AMTHREAD;
  GRANT CREATE SESSION TO AM4CICS;
  ```

- If you are not using AM4COID and AM4CAUTH, then the Oracle Access Manager control thread uses OS authenticated logon based on the CICS applid (for a local database) or CICS job name (for a remote database). Refer to the *Oracle Database Administrator's Guide* for more information about OS authenticated logins. The OS_AUTHENT_PREFIX initora parameter is prefixed to the CICS applid to create the Oracle userid. For example, if OS_AUTHENT_PREFIX = "" (null prefix), and if the CICS applid is CICSAPPL, then the following statements would be used to define Oracle Access Manager for CICS TS:

  ```
  CREATE USER CICSAPPL IDENTIFIED EXTERNALLY;
  GRANT CREATE SESSION TO CICSAPPL; ````````````
  ```

Note: If a local database is accessed and the CICS applid is used, it is important to review the LOGON_AUTH setting in the Database Service definition.

To enable FORCE and SELECT privileges for recovery processing if COMMIT is set to CICS (that is, if CICS SYNCPOINT processing is used instead of Oracle COMMIT), then you must issue the following statements, where *userid* is the Oracle Access Manager for CICS TS userid from above:

```
GRANT FORCE ANY TRANSACTION TO userid;
GRANT SELECT ON SYS.PENDING_TRANS$ TO userid;
GRANT SELECT ON SYS.PENDING_SESSIONS$ TO userid;
```

Step 5: Set INITORA Parameter and Prepare Host

You must complete the following steps on the server for Oracle Access Manager for CICS TS. For additional information, refer to the Oracle server installation documentation.

Step 5.1: Review IDLE_TIMEOUT Parameter

If the database server is a localbOSDI database service, then review the Database Region Parameter, IDLE_TIMEOUT. Do not use this parameter if PROTECT=YES is used or the TYPE=THREAD statement is in the thread table. If PROTECT=NO, ensure this setting is at least twice the value of CINTERVL in the TYPE=START statement.

Step 5.2: Set REMOTE_OS_AUTHENT

If you are preparing a remote host for Oracle Access Manager for CICS TS, and if OS authenticated logon is being used in step 4, then set the INITORA parameter REMOTE_OS_AUTHENT to TRUE on the remote database.

Step 5.3: Set OS_AUTHENT_PREFIX

If OS authenticated logon is being used, then ensure that the value of the INITORA parameter OS_AUTHENT_PREFIX on the remote database is used in the GRANT statements in "Step 4: Define CICS TS to Oracle and Grant Privileges".

Step 6: Generate the ORACSTUB Stub for CICS TS

To generate the ORACSTUB stub for CICS TS:

1. Determine your CICS TS adapter name.
2. Create the generation statement. Refer to SRCLIB library member ORACSTB for an example.
3. Generate ORACSTUB. Refer to SRCLIB library member ORACSJCL for an example.
4. Relink your Oracle Precompiler programs with the generated ORACSTUB. Place the library containing ORACSTUB in the SYSLIB concatenation of the link JCL and insure the SYSLIN DD contains an INCLUDE SYSLIB (ORACSTUB) statement.

Note: If you are using an Access Manager for CICS TS stub prior to release 9.2, you must regenerate the stub for CICS TS and relink applications with the new stub. Use of an older stub will result in an ORAP application abend.

Step 7: Update CICS TS Tables to Include Oracle Access Manager for CICS TS

If you are running CICS TS release 2.2 or less, then complete Steps 7.1 through 7.3.

If you are running CICS TS release 3.0 or higher, then do not perform Step 7. Instead, use the RDO definition statements in member ORACSD of the Oracle SRCLIB library as SYSIN for JCL to invoke DFHCSDUP. Before you submit the job, remove all comments. Change the appropriate data set names in the DD statements as appropriate for your site.

After successful completion, issue this RDO command to install the CICS tables:

```
CEDA INSTALL GROUP (ORACLE)
```

Proceed to "Step 8: Start Oracle Access Manager for CICS TS Adapter".

Step 7.1: Define Oracle Access Manager for CICS TS Programs to CICS

Before you can use Oracle Access Manager for CICS TS, you must define several resources to CICS by specifying CICS CEDA transactions.

Normally definitions for resources used by CICS, such as programs and transactions, are contained in GROUPS containing logically related resources. A GROUP is created when it is first specified in a CEDA DEFINE command. In these examples, the assigned GROUP name is ORACLE.

Issue these commands to define the programs used by Oracle Access Manager for CICS TS to CICS:

```
CEDA DEFINE PROGRAM(ORACICS) GROUP(ORACLE) LANGUAGE(ASSEMBLER)
CEDA DEFINE PROGRAM(CICADPX) GROUP(ORACLE) LANGUAGE(ASSEMBLER)
```

After executing each CEDA DEFINE command, CICS displays the message DEFINE SUCCESSFUL at the bottom of a full screen panel.

To end a current CEDA transaction before entering a new command, press the [PF3] and [Clear] keys.

Step 7.2: Define Oracle Access Manager for CICS TS Transactions to CICS

Once you have specified the programs, define the Oracle Access Manager for CICS TS control transaction with this command. Enter the command string on one line:

```
CEDA DEFINE TRANSACTION(xxx) GROUP(ORACLE) TWASIZE(0) PROGRAM(ORACICS)
```

where *xxx* is the name of the transaction you use as the Oracle Access Manager for CICS control transaction.

Note: The Oracle Access Manager control transaction also requires you to specify the TASKDATAKEY(CICS) parameter.

Step 7.3: Install Oracle Access Manager for CICS TS Resources

After you have defined the programs and transactions, install the newly created Oracle group. In this example, the group name is `ORACLE.` Use this command to install the new group:

```
CEDA INSTALL GROUP (ORACLE)
```

Step 8: Start Oracle Access Manager for CICS TS Adapter

To make Oracle Access Manager for CICS TS available to CICS TS transactions, you must start the Oracle Access Manager for CICS TS adapter. An adapter is a CICS TS program that interfaces between transactions and a system such as Oracle. To start the adapter, you can use:

```
transaction START MOD(module)
```

where:

Table 11–5 Variable Descriptions for Code Example

Variable	Description
module	is the name of the assembled thread definition table containing the Oracle Access Manager for CICS TS control transaction.
transaction	is the Oracle Access Manager for CICS TS control transaction.

For example, if the control transaction is ORA2 and the assembled thread definition table name is ORA0, you issue the command:

```
ORA2 START MOD(ORA0)
```

If the START command is successfully executed, the connection between CICS TS and Oracle is successfully established.

Step 9: Set Up Automatic Initialization for Oracle Access Manager for CICS TS

If you want to initialize Oracle Access Manager for CICS TS automatically at CICS initialization time and automatically shutdown Oracle Access Manager for CICS TS at CICS TS termination, then the Oracle Access Manager for CICS TS control program must be invoked during CICS TS startup and shutdown. This step is optional. You can run ORACICS at CICS TS initialization by using a CICS TS provided initialization parameter or by using the PLTPI facility. At CICS TS termination, you can run ORACICS by using of the PLTSD facility.

Step 9.1: Using the CICS TS initialization parameter

The CICS TS System Initialization Parameter (SIT) INITPARM can be used to run ORACICS. This can be in the SIT table or specified as a startup overide. The advantage of this method is that Access Manager for CICS is fully initialized before the "Control is being given to CICS" point.

For example:

```
INITPARM=(ORACICS='ORAC START MOD(ORA0)')
```

where ORAC is the Access Manager for CICS transaction code and ORA0 is the thread table name.

You can use any valid startup command.

Step 9.2: Using the PLTPI Program

The ORACICS macro generates a command level CICS TS program that is added to the CICS PLTPI table for automatic initialization during CICS TS startup. An example of the command to generate the PLTPI program is:

```
ORACICS TYPE=PLTPI,START=ORA0,TRANSAC=ORA2
```

In this example, ORA0 is the thread table definition and ORA2 is the Oracle Access Manager for CICS TS control transaction. This command is equivalent to issuing one of these commands from a terminal:

```
ORA2 START MOD(ORA0)
```

or

```
ORA2 START ORA0
```

The ORACICS TYPE=PLTPI statement is used as input to the assembly procedure and produces an assembler language CICS TS command-level program.

Use the following JCL to create the PLTPI program:

```
//ORAPLTPI JOB (ORA),'Oracle PLTPI PROGRAM'
//CIN      EXEC PGM=CINAMES,REGION=4000K
//*
//STEPLIB  DD DSN=ORACLE.V10G.CMDLOAD,DISP=SHR
//SYSIN    DD DUMMY
//SYSOUT  DD SYSOUT=*
//SYSERR  DD SYSOUT=*
//*  REPLACE INFILE WITH USER TNSNAMES SOURCE TO BE USED FOR CICS
//INFILE  DD DISP=SHR,DSN=ORACLE.ORA1.PARMLIB(TNSNAMES)
//OUTFILE  DD DSN=&&TEMPPDS(CINAMES),
//            UNIT=SYSDA,DISP=(,PASS),
//            DCB=(RECFM=FB,LRECL=80,BLKSIZE=400),
//            SPACE=(400,(100,100,5))
//ASM      EXEC PGM=IEV90,REGION=4000K
//SYSPRINT DD SYSOUT=*
//SYSLIB  DD DSN=&&TEMPPDS,DISP=(OLD,DELETE)
//        DD DSN=ORACLE.V10G.MACLIB,DISP=SHR
//        DD DSN=CICS.MACLIB,DISP=SHR
//        DD DSN=SYS1.MACLIB,DISP=SHR
//SYSUT1  DD UNIT=SYSDA,SPACE=(CYL,(5,5))
//SYSPUNCH   DD ***dataset for assembler pltpi source program
//SYSIN      DD *
          ORACICS TYPE=PLTPI,START=ORA0,TRANSAC=ORA2
/*
```

Step 9.3: Generate the PLTSD Program

The ORACICS macro generates a command-level program that is added to the CICS PLTSD table for automatic shutdown during CICS termination. An example of the command to generate the PLTSD program is:

```
ORACICS TYPE=PLTSD,PROGRAM=ORACICS,STOP=ORA0
```

In this example, `ORACICS` is the name of the Oracle Access Manager for CICS TS control program and ORA0 is the name of the CICS TS adapter to be stopped. This command is equivalent to issuing the terminal command:

```
ORA2 STOP NAME(ORA0)
```

The ORACICS TYPE=PLTSD statement is used as input to the assembly procedure and produces an assembler language CICS TS command-level program.

Use the following JCL to create the PLTSD program:

```
//ORAPLTSD JOB (ORA),'Oracle PLTSD PROGRAM'
//CIN      EXEC PGM=CINAMES,REGION=4000K
//*
//STEPLIB  DD DSN=ORACLE.V10G.CMDLOAD,DISP=SHR
//SYSIN    DD DUMMY
//SYSOUT  DD SYSOUT=*
//SYSERR  DD SYSOUT=*
//*  REPLACE INFILE WITH USER TNSNAMES SOURCE TO BE USED FOR CICS
//INFILE  DD DISP=SHR,DSN=ORACLE.ORA1.PARMLIB(TNSNAMES)
//OUTFILE  DD DSN=&&TEMPPDS(CINAMES),
//            UNIT=SYSDA,DISP=(,PASS),
//            DCB=(RECFM=FB,LRECL=80,BLKSIZE=400),
//            SPACE=(400,(100,100,5))
//ASM      EXEC PGM=IEV90,REGION=4000K
//SYSPRINT DD SYSOUT=*
//SYSLIB  DD DSN=&&TEMPPDS,DISP=(OLD,DELETE)
//        DD DSN=ORACLE.V10G.MACLIB,DISP=SHR
//        DD DSN=CICS.MACLIB,DISP=SHR
//        DD DSN=SYS1.MACLIB,DISP=SHR
//SYSUT1  DD UNIT=SYSDA,SPACE=(CYL,(5,5))
//SYSPUNCH   DD ***dataset for assembler pltsd source program
//SYSIN      DD *
          ORACICS TYPE=PLTSD,PROGRAM=ORACICS,STOP=ORA0
/*
```

Step 9.4: Input PLTSD and PLTPI to the CICS DFHEITAL Procedure

Use the generated assembler language CICS TS command-level programs (PLTSD and PLTPI) as input to the CICS DFHEITAL procedure.

Step 9.5: Add the User-Defined Name to the PLTPI Table

Add the user-defined name of the load module containing the generated program for PLTPI (output of the DFHEITAL procedure) to the PLTPI table. If your output from DFHEITAL is named ORAPLTI, then you can also name your load module ORAPLTI. For example:

```
CEDA DEFINE PROGRAM(ORAPLTI) GROUP(ORACLE) LANGUAGE(ASSEMBLER)
```

Define the load module to CICS TS with a CEDA DEFINE command. The generated program can be modified to include override parameters.

This program must be placed after PROGRAM=DFHDELIM in the CICS PLTPI table.

Step 9.6: Add the User-Defined Name to the PLTSD Table

Add the user-defined name of the load module containing the generated program for PLTSD (output of the DFHEITAL procedure) to the CICS PLTSD table. If your output from DFHEITAL is named ORAPLTS, then you can name your load module ORAPLTS. For example:

```
CEDA DEFINE PROGRAM(ORAPLTS) GROUP(ORACLE) LANGUAGE(ASSEMBLER)
```

Define the load module as a program to CICS TS with a CEDA DEFINE command.

Step 9.7: Make Generated Programs Available

Ensure the generated programs are available in a library included in the CICS DFHRPL.

While using the PLTPI and PLTSD programs, status messages are issued to the system console.

Post-Configuration Steps

The following steps are optional.

Step 1: Modify the Sample Compilation Procedures

The Oracle SRCLIB library contains these procedures for preparing high-level language programs. Modify these procedure names according to your site's naming conventions and move them to an appropriate procedure library if necessary.

Table 11–6 Compilation Procedures and Libraries

Procedure	Library
CICSPCCC	for COBOL programs
CICSPCCI	for IBM C/370 programs

Step 2: Use the SRCLIB Member OSAMPLE

SRCLIB member OSAMPLE is a COBOL II program that displays an employee name from the EMP table. To use the sample program, you must:

1. Ensure the EMP table is on the database.
2. Ensure SCOTT/TIGER is a valid logon id on the database. Also ensure SCOTT /TIGER has access to the EMP table.
3. Ensure "Step 6: Generate the ORACSTUB Stub for CICS TS", is completed.
4. Tailor the JCL in SRCLIB member CICSPCC2 for your installation. Then execute CICSPCC2 to compile OSAMPLE. This places OSAMPLE into a library in the CICS DFHRPL concatenation.
5. Define the program OSAMPLE to CICS TS by using standard RDO procedures.
6. Define a transaction to CICS TS (for example, OSAM) using standard RDO procedures and associate it with the OSAMPLE program.
7. Once the START command is issued for Oracle Access Manager for CICS TS, you can invoke the transaction.

Multiple Versions in the Same CICS TS Region

This version of Oracle Access Manager for CICS TS can coexist in the same CICS TS region with a version prior to release 9.2. For example, release 10.1 can co-exist with release 9.0.x or earlier, but not with a release 9.2.x.

To configure a second version, perform the following steps:

1. (This step is required only if both versions will be executing concurrently. If not, go to Step 2.) If both versions will be executing concurrently, create an ORACSTUB stub specifying the adapter name that has been chosen for the 10.1 version. This will be linked with applications programs accessing the 10.1 version.
2. This version uses the ORACICS load module. If a prior version uses the ORACICN load module name as distributed, there is no conflict.
3. Define a second control transaction to CICS TS, associating this transaction with the ORACICS program.

Recovery Considerations

Resource recovery is determined by whether COMMIT (CICS) or COMMIT (Oracle) was designated to Oracle Access Manager for CICS TS as the recovery choice in the thread definition table parameter COMMIT.

Using COMMIT (CICS)

CICS TS maintains and recovers its own protected resources (that is, VSAM data sets, DL/I databases, and so forth) with the use of the SYNCPOINT and ROLLBACK commands. Use SYNCPOINT to indicate a logical unit of work is complete and does not need to be backed out in case of failure. Use ROLLBACK to indicate a logical unit of work is incomplete and changes need to be backed out in case of failure.

When a CICS TS transaction syncpoints explicitly by issuing an EXEC CICS SYNCPOINT or implicitly at transaction end, Oracle Access Manager for CICS TS ensures all Oracle server updates in the logical unit of work (LUW) are committed or backed out. Use the Oracle two-phase commit support to accomplish this.

A transaction abend, ORAP, error occurs if you specify the CICS option and an Oracle EXEC SQL COMMIT or EXEC SQL ROLLBACK is issued.

Using COMMIT (Oracle)

Oracle has its own recovery mechanism, independent of CICS TS, and utilizes the COMMIT and ROLLBACK SQL commands to control it. To commit or rollback Oracle changes, an application must issue an EXEC SQL COMMIT or EXEC SQL ROLLBACK. If an application does not issue a COMMIT or ROLLBACK SQL command, then all outstanding Oracle database updates are rolled back at the end of the CICS TS transaction.

A CICS SYNCPOINT also does not perform a SQL COMMIT, nor does a SQL COMMIT perform a CICS SYNCPOINT. If an application is performing update operations on both CICS protected resources and Oracle tables, then the application must issue both a CICS SYNCPOINT and a SQL COMMIT to affect both sets of resources.

Two-Phase Commit Processing under CICS TS

When CICS syncpoint processing is selected as the resource commit or recovery choice, Oracle Access Manager for CICS TS can participate in two-phase commit processing under CICS TS.

CICS TS serves as the coordinator and Oracle Access Manager for CICS TS uses the two-phase commit protocol to interface with the Oracle server.

This process is transparent to the transaction. Oracle Access Manager for CICS TS implements the CICS syncpoint resource manager interface, enabling Oracle resources to participate in CICS distributed transactions and removing the requirement that Oracle COMMIT and ROLLBACK commands be issued in addition to the CICS SYNCPOINT command.

To use the CICS syncpoint resource manager interface, you must specify CICS as the recovery choice in the thread table or as a startup option. You can specify the option by:

- Using the COMMIT thread table parameter

  For more information, refer to the "Thread Definition Table Parameters" on page 11-5.

- Using the COMMIT parameter on the START command

  For more information, refer to the START command description on page 11-34.

First Phase

In the first phase of the two-phase commit protocol, the CICS syncpoint manager presents a PREPARE request, which requests a promise to commit or rollback the transaction to Oracle Access Manager for CICS TS. Oracle Access Manager for CICS TS communicates with the Oracle servers and returns one of three responses to the CICS syncpoint manager:

Table 11–7 Responses to CICS Syncpoint Manager

Response	Description
ABORT	indicates the Oracle server could not complete the CICS PREPARE request.
PREPARED	indicates the Oracle server has all resources necessary to subsequently commit and rollback the transaction. When the prepare phase is completed, the transaction is said to be in-doubt.
READ-ONLY	indicates no data has been modified so no **PREPARE** is necessary.

Second Phase

The second phase of two-phase commit processing is the commit phase. The CICS syncpoint manager presents a COMMIT/ROLLBACK request to Oracle Access Manager for CICS TS. Oracle Access Manager for CICS TS then communicates with the Oracle server to complete the processing.

Failures can result during two-phase commit processing. These items are in-doubt resolutions:

- CICS TS warm or emergency restart
- Oracle server restart
- Manual recovery

CICS TS Warm or Emergency Restart

In this case, CICS TS has access to a log of in-doubt LUWs. When Oracle Access Manager for CICS TS starts, it resynchronizes with CICS TS and the Oracle server to obtain the transactions that are in-doubt, and communicates with the Oracle server to complete two-phase commit processing.

LUWs that include Oracle and other CICS TS recoverable resources can be lost in the case of a CICS TS cold start and might require manual resolution.

Oracle Server Restart

When Oracle Access Manager for CICS TS detects an Oracle server restart, it resynchronizes in-doubt Oracle server LUWs with CICS TS and communicates with the Oracle server to complete two-phase commit processing.

Manual Recovery

Oracle in-doubt CICS TS transactions should not be manually committed or rolled back using the FORCE command. However, some situations, such as a CICS TS cold start, might require manual intervention. The Oracle server maintains a pending transaction view, DBA_2PC_PENDING. The GLOBAL_TRAN_ID in DBA_2PC_PENDING for CICS TS transactions is the concatenation of these fields in the order shown:

Table 11–8 Concatenated Fields for GLOBAL_TRAN_ID

Field Name	Description
`OPS$applid`	is the eight character CICS applid, or the value of AM4COID in the thread table.
`adapter name`	is the four-character name of an Oracle Access Manager for CICS TS adapter.
`logical unit of work`	is eight characters and represents the value of the logical unit of work obtained from CICS TS.

For more information about the `OPS$applid` and `adapter name` fields, refer to "Configuration Steps" in this chapter on page 11-4. For additional information about using DBA_2PC_PENDING for manual recovery, refer to *Oracle Database Administrator's Guide*.

Shutting Down Oracle Access Manager for CICS TS with FORCE

To shut down Oracle Access Manager for CICS TS forcibly, use the STOP command with the IMMEDIATE FORCE parameter. When the IMMEDIATE FORCE parameter is used with the STOP command, Oracle Access Manager for CICS TS terminates all threads and shuts down. Caution must be used with this parameter. The shutdown is not an orderly shutdown. For more information, refer to the STOP command on page 11-35.

If you start Oracle Access Manager for CICS TS after you start the Oracle server, then automatic restart takes effect when Oracle terminates. When the Oracle server is restarted, Oracle Access Manager for CICS TS automatically restarts and resynchronizes any in-doubt logical units of work. If you start Oracle Access Manager for CICS TS before the Oracle server, then you must perform the initial start of Oracle Access Manager for CICS TS manually.

CEDF Support

You can use CICS EDF to debug Oracle Access Manager by transaction. SQL statements are shown on the EDF screen before and after each transaction is run. To begin an EDF session, start the adapter and include the keyword EDF. For example:

```
ORA2 START MOD(ORA0) EDF
```

A connect statement will only display as "connect." There is no userid and password information displayed.

SQL statements are displayed as coded; the values for variables are not resolved.

Oracle Access Manager for CICS TS V$SESSION Information

Oracle Access Manager for CICS TS uses the PL/SQL package `dbms_application_info` to maintain the following fields in the V$SESSION table. Ssession information is updated for each CICS TS transaction when the database is first accessed:

- The Module field contains the initial application program name.
- The Client_info field contains 16 bytes of information, as follows:

```
CICS transaction code(4) column 1-4
CICS terminal id(4)      column 5-8
CICS userid(8)           column 9-16
```

Oracle Access Manager for CICS TS Recovery and Autorestart

The following are three scenarios for determining whether Oracle Access Manager for CICS TS will restart automatically or will need to be restarted manually in the event that the database becomes unavailable:

- If Oracle Access Manager for CICS TS is in the process of starting up or shutting down and the database becomes unavailable, Oracle Access Manager for CICS TS will need to be restarted manually.
- If Oracle Access Manager for CICS TS is running and it detects that the database is no longer available, it performs an emergency shutdown and issues message CIC-00026.

  Transactions that are executing receive an ORAP abend or a CICS ASP7 abend, depending on their execution point. New transactions receive an ORAP abend or a CICS AEY9 abend or AETA abend.

  Oracle Access Manager for CICS TS will monitor the database and when it becomes available, will automatically restart and issue message CIC-00007.
- If Oracle Access Manager for CICS TS is accessing a local database and the database service is either stopped, cancelled, or otherwise becomes unavailable due to an uncontrolled shutdown, the Oracle Access Manager for CICS TS will restart automatically under the control of a retry limit and a time interval.

  The retry limit is the maximum number of times a loss of service condition will trigger automatic restart. To deactivate the mechanism, change the retry limit to 0. The default is 2 (for example, after 2 loss of OSDI service conditions have occurred, manual restart is required). If the limit is reached and another loss of service occurs, a CIC-99999 message stating that manual restart is required will immediately precede the CIC-00026 emergency shutdown message.

The time interval, in the form of *hhmmss*, is the time to wait before first restart attempt when an OSDI loss of service occurs. Note that this time interval only applies in this situation. If the service is not available after this time interval, Oracle Access Manager for CICS TS continues to attempt a restart, but the time interval reverts to the normal, CINTERVAL value from the thread table. The default is 30 seconds.

A new display, similar to other Oracle Access Manager for CICS TS transaction displays, is provided to update the retry and interval fields. For example, if you enter `ORA2 ALTER NAME (ORA0)`, where `ORA2` is the control transaction, and `ORA0` is the adapter name, you receive an ORA0 Alter display screen formatted with the current retry and interval values.

The values can be modified by typing over them. Pressing the ENTER key updates and redisplays the values so they can be verified.

Note: The values can only be changed when the adapter is active. Therefore they cannot be modified when the restart mechanism is in progress.

Pressing thePF3 key exits the Alter display The values are reset to the defaults after a normal shutdown or startup of Oracle Access Manager for CICS TS.

Oracle Access Manager for CICS TS Command Usage

Oracle Access Manager for CICS TS commands are entered with the transaction id followed by the Oracle Access Manager for CICS TS command. The general format is:

```
trans_id command parms
```

where:

Table 11–9 Variable Description for Code Example

Variable	Description
`trans_id`	is the Oracle Access Manager for CICS TS transaction id (usually `ORA2`).
`command`	is the Oracle Access Manager for CICS TS command.
`parms`	is one or more parameters for the command.

Oracle Access Manager for CICS TS commands can be entered in two ways:

- You can enter the CICS transaction id (usually `ORA2`) followed by the command. For example:

  ```
  ORA2 DISPLAY NAME(ORA0)
  ```

  In this case, the control transaction sends the display to the terminal.

- You can enter only the transaction id for the control transaction. In this case, the control transaction prompts you for a command.

  Use the [PF3] key to exit from the control transaction.

DISPLAY

The following section contains information about DISPLAY.

Options

DISPLAY NAME, DISPLAY TRAN NAME, DISPLAY STATUS NAME

Syntax

```
DISPLAY NAME(adapter)
DISPLAY TRAN() NAME(adapter)
DISPLAY STATUS NAME(adapter)
```

where `adapter` is the name of an Oracle Access Manager for CICS TS adapter.

Purpose

This command allows you to monitor your Oracle Access Manager for CICS TS system. The screen displayed varies according to which DISPLAY command is issued.

DISPLAY NAME Example

DISPLAY NAME displays a screen showing general information about an Oracle Access Manager for CICS TS system.

When the command:

```
ORA2 DISPLAY NAME(ORA0)
```

is issued, where `ORA2` is the name of the transaction defined to control Oracle Access Manager for CICS TS and `ORA0` is the name of the Oracle Access Manager for CICS TS adapter, the screen is displayed:

DISPLAY TRAN NAME Example

DISPLAY TRAN NAME displays a screen showing all the transactions defined in a thread definition table.

When the command:

```
ORA2 DISPLAY TRAN() NAME(ORA0)
```

is issued, where ORA2 is the name of the transaction defined to control Oracle Access Manager for CICS TS, TRAN()is the transaction identifier and ORA0 is the name of the Oracle Access Manager for CICS TS adapter, the screen is displayed.

You can specify the TRAN() parameter as follows:

```
TRAN - default all transactions
TRNA(ZZZZ) - ZZZZ transaction
```

You can also use wildcards with the TRAN() parameter. For example:

```
TRAN(A*)
TRAN(*99)
TRAN(C*99)
```

Pressing the [Enter] key refreshes the display, pressing the [PF7] key scrolls backward, and pressing the [PF8] key scrolls forward.

The status codes and the thread characteristics they indicate for the DISPLAY TRAN NAME display are:

Table 11–10 Status Codes and Thread Characteristics

Status Code	Thread Characteristic
20	reserved
40	use general pool thread (WAIT=POOL)
80	wait for available thread (WAIT=YES)

The status code can contain more than one meaningful value. For example, a value of 60 (40 + 20) indicates transactions use threads, are running on worker tasks, and if no threads are available, a pool thread is used.

You can display all the active threads used for a particular transaction by entering **S** to the left of a transaction displayed in the DISPLAY TRAN NAME display:

Pressing the [Enter] key refreshes the display, pressing the [PF7] key scrolls backward, and pressing the [PF8] key scrolls forward.

In this display, the value `1` in the `EXEC SQL` column indicates one SQL call has been performed. The columns in this display are:

Table 11–11 Columns in Display

Column	Description
`THREAD`	thread number
`TASK`	CICS task number
`TERM`	CICS terminal id
`PROGRAM`	program name
`EXEC SQL`	number of SQL calls performed
`COMMIT`	number of SQL commits performed if using ORACLE COMMIT
`ROLLBACK`	number of SQL rollback calls performed if using ORACLE ROLLBACK
`ERROR`	current error code value

DISPLAY STATUS NAME Example

DISPLAY STATUS NAME displays a screen showing all active threads. An active thread is one currently in use by a CICS TS transaction.

When the command:

```
ORA2 DISPLAY STATUS NAME(ORA0)
```

is issued (where ORA2 is the name of the transaction defined to control Oracle Access Manager for CICS TS and ORA0 is the name of the Oracle Access Manager for CICS TS adapter), the screen is displayed

Pressing the [Enter] key refreshes the display, pressing the [PF7] key scrolls backward, and pressing the [PF8] key scrolls forward.

The columns in this display are:

Table 11–12 Columns in Display

Column	Desciption
`THREAD`	thread number
`TASK`	CICS task number
`TRAN`	CICS transaction id
`TERM`	CICS terminal id
`PROGRAM`	program name
`LOGONID`	thread user's logon id
`STATUS`	status code

The status codes and the thread characteristics they indicate for the DISPLAY STATUS NAME display are:

Table 11–13 Status Codes and Thread Characteristics

Status Code	Thread Characteristic
00	no values set
08	reserved
20	high priority thread (for subtasks only, PRIORITY set to HIGH)
40	pool thread
80	protected

Note: All the status codes except code 20 apply to the display of both local and remote Oracle data. Status code 20 applies only to local Oracle data.

The status code can contain more than one meaningful value. Some codes are not listed and can be ignored because they are for diagnostic purposes.

To purge a transaction or thread from within the DISPLAY STATUS NAME display, enter **P** to the left of the transaction or thread. You receive one of the following responses indicating the outcome of the purge operation:

Table 11–14 Outcome of Purge Operation

Response	Description
*	indicates the purge was successful.
E	indicates the purge was not successful.
?	indicates the current transaction was completed before the purge was attempted.

Caution: The purge facility performs cleanup, and issues an ORAP application abend for the transaction.

START

This section contains information about START.

Syntax

```
START MOD(modname) [MAX(threads) SSN(ssn) NAME(adapter) COMMIT(option)]
```

where:

Table 11–15 Variable Descriptions for Code Example

Variable	Description
`modname`	is the name of the load module for the Oracle Access Manager for CICS thread definition table. If the NAME parameter is not specified, then this is used as the name of the Oracle Access Manager for CICS TS adapter.
`threads`	is the maximum number of threads supported by this adapter. This overrides the value specified in the ORACICS macro.
`ssn`	is the name of the Oracle subsystem corresponding to this adapter, if the Oracle Access Manager for CICS TS transaction is accessing local Oracle data. If the transaction is accessing remote Oracle data, then the `ssn` is the four-character alias name used in the CICS TNSNAMES entry.
`adapter`	is the name of the CICS TS adapter. If this parameter is not specified, then the `modname` is used as the name for the CICS TS adapter.

Table 11–15 (Cont.) Variable Descriptions for Code Example

Variable	Description
`option`	is the recovery choice. This overrides the value specified in the ORACICS macro. The two valid values for `option` are: CICS to use CICS SYNCPOINT ORACLE to use Oracle COMMIT/ROLLBACK

Purpose

This command starts the Oracle Access Manager for CICS TS adapter.

The parameter values specified in the START command override any values specified in the load module named in the MOD parameter.

Because the load module specified in the MOD parameter contains thread definitions, starting the adapter with only the MOD parameter is normally sufficient.

START Example

This example provides only the load module and adapter names:

```
ORA2 START MOD(ORA0) NAME(ORA0)
```

where:

Table 11–16 Variable Descriptions for Code Example

Variable	Description
`ORA2`	is the Oracle Access Manager for CICS TS control transaction
`ORA0`	is the name of the load module
`ORA0`	is the name of the thread definition table. Subsystem name and thread definitions are taken from the `ORA0` table.

In this example, an override is specified for the Oracle Access Manager for CICS TS adapter name.

STOP

This section contains information about STOR.

Syntax

```
STOP NAME(adapter) [IMMEDIATE] [FORCE] [WAIT]
```

where:

Table 11–17 Variable Descriptions for Code Example

Variable	Description
`adapter`	is the name of an Oracle Access Manager for CICS TS adapter started with the START command.
`IMMEDIATE`	is an optional parameter that rejects all requests but does not shut down until the system quiesces.
`FORCE`	is an optional parameter that terminates all threads and shuts down. Any active transactions will receive an ORAP application abend on the next Oracle access. When there are no active Oracle transactions, the adapter will shut down.
`WAIT`	is an optional parameter that waits for the adapter to shut down before returning control to the terminal.

Purpose

This command stops an Oracle Access Manager for CICS TS adapter started with the START command. When the FORCE option is used with the STOP command, Oracle Access Manager for CICS TS abends all currently running transactions and forcibly shuts down.

STOP FORCE Example

This example shows the STOP command with IMMEDIATE FORCE:

```
ORA2 STOP NAME(ORA0) IMMEDIATE FORCE
```

where ORA2 is the Oracle Access Manager for CICS TS control transaction and ORA0 is the Oracle Access Manager for CICS TS adapter name.

Oracle Access Manager for CICS TS Storage Requirements

This section describes the storage requirements for Oracle Access Manager for CICS TS.

Base Code Storage Requirements

Table 11–18 lists the base code storage requirements for Oracle Access Manager for CICS TS.

Table 11–18 Oracle Access Manager for CICS TS Base Code Storage Requirements for Configuration

Usage	Below 16M	Above 16M
ORACICS	0 bytes	28K
CICADPX	24 bytes	0K
LIBCLNTS	0 bytes	24M[1]

[1] This is above-the-line storage, not managed by CICS TS.

Adapter Storage Requirements

Table 11–19 lists the adapter storage requirements for Oracle Access Manager for CICS TS, which are the same for both the local and remote configurations. The total variable storage requirement per active Oracle server/CICS TS transaction is 30,748 bytes.

The following table displays the Oracle Access Manager for CICS TS adapter storage requirements for local and remote configurations:

Table 11–19 Adapter Storage Requirements for Local and Remote Configurations

Type of Storage	Usage	Area	Below 16M	Above 16M
Fixed	Per active Access Manager for CICS TS adapter	Global Exit Area	1300 bytes	0 bytes
Variable	Per active Oracle Database /CICS TS transaction	Local Exit Area	432 bytes	0 bytes

Thread Table Storage Requirements

For the thread table, a fixed amount of storage is used. This calculation is used to determine a thread table's fixed storage requirements (located above the 16M line):

```
8176 bytes + (1096 bytes x MAXTHRDS)
```

Use the same calculation for both the local and remote configurations.

Connected Thread Storage Requirements

Table 11–20 lists the connected thread storage requirements for Oracle Access Manager for CICS TS.

Table 11–20 Oracle Access Manager for CICS TS Connected Thread Storage Requirements

Type of Storage	Usage	Area	Below 16M	Above 16M
Variable	Per active Oracle Database /CICS TS connection	Oracle client storage	0 bytes	100K bytes[1]

[1] This is above-the-line storage, not managed by CICS TS.

All buffer storage for connected threads under CICS TS release 4.1 or higher is obtained above the 16M line.

12 Oracle Access Manager for IMS TM

This chapter discusses how to configure and operate the Oracle Access Manager for IMS TM. It also describes how the product is integrated with IMS.

The following topics are included:

Overview

Oracle Access Manager for IMS TM provides the interface between Oracle Precompiler programs (COBOL, C, or PL/I languages) executing Oracle SQL in an IMS TM transaction, and an Oracle database. Oracle Access Manager for IMS TM communicates with an OSDI local database server or it communicates with a remote Oracle database. The target services, or remote database, is defined in the Resource Translation Table (RTT) discussed in this chapter. Each instance of Oracle

Access Manager for IMS TM communicates directly with one OSDI server or one remote database server.

Connections for Oracle IMS transactions, whether to a local or remote database, are created and revised based on criteria defined in the RTT. When the IMS TM transaction requests that work be done in the Oracle database, Oracle Access Manager utilizes the z/OS Oracle client code, LIBCLNTS, in an LE enclave prepared for this purpose. The Oracle client code communicates with a local Oracle server through cross-memory services or with a remote server through Oracle networking socket calls to satisfy the request. Starting with Oracle9*i* release 2, this direct use of socket calls eliminates the requirement for a Net service in earlier releases.

Oracle Access Manager for IMS TM Applications

You can run applications built with the following Oracle products for z/OS under Oracle Access Manager for IMS TM:

- Pro*COBOL
- Pro*C
- Pro*PL/1

Refer to *Oracle Database User's Guide for IBM z/OS* for additional information about using Oracle Precompilers and Oracle Call Interface.

Integration with IMS

Oracle Access Manager for IMS TM is based on the IMS External Subsystem Attachment Facility (ESAF), a published IMS interface for incorporating external resources into IMS transaction management. ESAF defines program interfaces (exits) that IMS invokes for:

- Initialization and connection
- Signon and signoff
- Thread creation
- Synchronization
- Termination functions

Some aspects of Oracle Access Manager for IMS TM installation, configuration, and operation are specific to the ESAF architecture. This guide provides some information on ESAF; however, IBM IMS documentation is the definitive source for ESAF information.

Although ESAF is named and discussed as subsystems, Oracle Access Manager for IMS TM does not run as a subsystem in a separate address space. When Oracle Access Manager for IMS TM is configured, IMS region initialization loads the main body of Oracle Access Manager for IMS TM code into the control region and into each dependent region accessing an Oracle database. IMS calls the various ESAF exits under the application management task (dependent regions) or the ESI task (control region). No additional tasks are created by or for Oracle Access Manager for IMS TM. Processing in the control region is primarily for recovery activities. Processing in the dependent region is for Oracle requests issued by transactions running in the region. The control region must connect to Oracle successfully before any dependent region is allowed to connect.

Oracle Access Manager for IMS TM Design

Oracle Access Manager for IMS TM minimizes resource consumption for processor utilization, memory utilization, and activities that can be serialized. One connection is maintained between an IMS region and a given target Oracle database, regardless of the number of distinct transactions or security contexts (Oracle user ids) the region hosts over time. Oracle Access Manager for IMS TM logically maps the combination of IMS thread and Oracle user id to the generic Oracle server session mechanism. The sessions are created and managed dynamically. They are based on the stream of transactions created by users and configuration parameters set by the installation.

The Oracle Access Manager for IMS TM code runs in 31-bit addressing mode and resides above the 16M address line. All dynamically acquired virtual memory is also above the 16M line. Oracle Access Manager for IMS TM is completely reentrant. If the installation prefers, all IMS regions accessing Oracle databases can share a single copy of the code. Each instance of Oracle Access Manager for IMS TM (that is, each distinct Oracle database to be accessed) requires allocation of less than 4K of global (ECSA) virtual memory.

Configuration Overview

You must install Oracle Database for z/OS including the Oracle Access Manager successfully before you configure the product by:

- Placing the main Oracle Access Manager for IMS TM code and supporting modules into the IMS RESLIB or another data set concatenated to RESLIB in the IMS region JCL
- Ensuring Oracle Access Manager for IMS TM code and supporting modules are available to STEPLIB or ORAAMIDD and DFSESL DD statements
- Installing the linking stub module AMILS (used to link Oracle Access Manager for IMS TM client transaction programs) in a load library accessible to application developers
- Placing Oracle Access Manager for IMS TM macros (used to define the product configuration) in an appropriate Assembler language macro source code library
- Installing the sample programs and material related to installation verification (this material is not required to configure the product)

These steps are specific to Oracle Access Manager for IMS TM. The installation process can also involve other components of Oracle Database for z/OS or Oracle8*i* packages.

You must configure an instance of Oracle Access Manager for IMS TM for each distinct Oracle database that is accessed by transactions in an IMS subsystem. IMS (ESAF) requires two identifying characteristics for each instance:

- Subsystem identifier (SSM)
- Language interface token (LIT)

Each is a one-character to four-character identifier you choose. The subsystem identifier is specific to Oracle Access Manager for IMS TM. It is not the subsystem identifier associated with the OSDI subsystem.

The LIT and SSM

The LIT is embedded in transaction programs using an instance of Oracle Access Manager for IMS TM. When a transaction program runs, IMS associates its LIT with a given subsystem identifier through an entry in the IMS subsystem parameter file, referred to as an SSM member of IMS PROCLIB. The SSM member is an IMS region parameter file defining each external subsystem that can be accessed from that region.

Different IMS regions can have different SSM members. The control region SSM must specify every external subsystem instance (Oracle Access Manager for IMS TM, DB2 for z/OS, or others) that the IMS subsystem can access. Dependent regions can access any external subsystem defined in the control region SSM by default. To limit the external subsystems available to a dependent region or vary the parameters for one or more subsystems from what is specified for the control region, create a separate SSM for the dependent region. To prevent access to all external subsystems from a region, specify a dummy (empty) SSM. You are responsible for ensuring that each dependent region SSM allows access to the external subsystems required by the transactions scheduled in that region.

Resource Translation Table

IMS designates the data that can be coded in an SSM entry. The few defined parameters are not sufficient for implementation of Oracle Access Manager for IMS TM. The SSM does allow you to specify an additional subsystem parameter module called a resource translation table (RTT) whose contents are not specified by IMS. IMS loads the module during region initialization and passes its address to Oracle Access Manager for IMS TM as part of the ESAF exit interface.

Oracle Access Manager for IMS TM uses the RTT to specify most of its parameters and options. At least one RTT is required for each Oracle Access Manager for IMS TM instance. You generate an RTT by coding S/370 Assembler language macros (examples provided with Oracle Access Manager for IMS TM) that create the needed parameter structures. These structures are queried by Oracle Access Manager for IMS TM at runtime.

Although the RTT is a generic ESAF entity, its contents are specific to the external subsystem being accessed. Therefore, the Oracle RTT bears no resemblance to the one used by DB2, for example, and serves a different purpose.

The installation codes macro parameters that specify:

- The Oracle Net address of the target database
- Characteristics of IMS transactions
- How target database sessions are managed
- Other details

The macro calls are assembled and linked to produce a load module. The load module is placed in (or concatenated to) the IMS RESLIB. The module’s name must be specified as the RTT parameter for the Oracle Access Manager for IMS TM instance in the control region SSM. For additional details, refer to "Configuration Steps" on page 12-13.

Different RTT modules can be created for the same target Oracle database to vary Oracle Access Manager for IMS TM behavior from region to region. The RTT specified in the control region SSM entry for the database is considered the master

RTT. It specifies parameters that cannot be varied among the dependent regions, such as the address of the target Oracle database. All of its parameters apply in dependent regions that do not have a separate RTT. The additional RTT modules are specified in the dependent region SSM.

Distributed Option Considerations

If the target Oracle database does not have the distributed option installed, then IMS transactions are not allowed to update any data in that database.

Security Considerations

Oracle and IMS/MVS security differ in some respects. Oracle database user ids can be up to 30 characters. User ids on z/OS are limited to seven or eight characters. You have a choice of security schemes when creating user ids for Oracle Access Manager for IMS TM. You can:

- Create new Oracle user ids specifically for Oracle Access Manager for IMS TM-based applications
- Replicate the z/OS user ids on the Oracle side and adhere to a single user id view
- Use existing Oracle user ids or schema names that differ from what is being used in z/OS

Determining the Oracle User id

Oracle Access Manager for IMS TM accommodates these differences by providing a flexible way to determine the Oracle user id for a given transaction. Oracle user ids are controlled by parameters generated in the RTT. In the RTT, you can specify the type of Oracle user id an IMS application can have:

1. The IMS user id

   For terminal-oriented transactions, this is the signon id if the terminal is signed on. Otherwise it is the IMS LTERM name. For non-message driven batch message processing (BMP), the IMS user id is one of a hierarchy of choices defined by IMS. In many IMS installations, the IMS user id is authenticated by RACF or a similar security product.

2. The IMS program specification program (PSB) name

   The IMS PSB name identifies the IMS application.

3. The application program (load module) name

4. A specific (constant) Oracle user id up to 30 characters

   This choice makes it possible for Oracle Access Manager for IMS TM to accommodate use of any Oracle user id.

If you want to replicate the IMS/MVS user id structure in Oracle, then choose the IMS user id. You can also choose the IMS PSB name and the application program if they fit your installation. You can specify choices on a PSB-by-PSB basis in the RTT generation, and you can designate a default choice for PSBs not explicitly specified.

Session Authentication

An Oracle session created for an IMS transaction using Oracle Access Manager for IMS TM is subject to Oracle's normal authentication mechanisms. These mechanisms are generally controlled by the Oracle database administrator or by system security staff. Oracle supports two kinds of session authentication:

- Password

  Password authentication requires the client (Oracle Access Manager for IMS TM in this case) to supply a password when establishing an Oracle session for a user id.

- EXTERNAL

  EXTERNAL authentication requires the client interface software to verify the user is already authenticated by the client operating environment.

A given Oracle user id is subject to one or the other of the authentication mechanisms, as specified by the DBA or security staff in the CREATE USER or ALTER USER SQL statement. In addition to individual user id specifics, the DBA can specify whether the Oracle instance allows EXTERNAL authentication of sessions coming through Oracle Net. If remote clients (including Oracle Access Manager for IMS TM) are prohibited, then they can use only password-authenticated user ids.

Without this restriction, Oracle Access Manager for IMS TM supports both forms of authentication. It specifies actual Oracle passwords for user ids and supports the client mechanics of EXTERNAL authentication when required by the intended Oracle user id. Oracle Access Manager for IMS TM does not invoke the z/OS SAF/RACF interface to authenticate the user id before using it with Oracle EXTERNAL authentication. The validity of the user id is an installation responsibility.

Oracle Access Manager for IMS TM does not automatically know an Oracle user id authentication mechanism or password. You specify these on a individual user id basis in the Oracle Access Manager for IMS TM RTT generation. You can provide a default choice for all unspecified user ids.

Oracle Access Manager for IMS TM does not store Oracle passwords in clear text. Oracle server release 7.1 and above transmits logon data (user id and possible password) to the target database in encrypted form.

Error Processing

IMS defines a characteristic called a region error option (REO), which specifies how an ESAF implementation handles non-application processing errors, such as a loss of communications with the external subsystem. The REO has one of three one-character values:

Table 12–1 REO Values

Value	Description
R	specifies error code associated with the failure are returned to the application program
Q	specifies the application program is terminated with an abend code U3044. The input transaction is requeued to be processed when access to the subsystem is restored.
A	specifies the application program is terminated with an abend code U3983. The input transaction is discarded. Oracle Access Manager for IMS TM supports all three REO processing options. IMS allows you to specify the REO at the region level in the IMS PROCLIB member defining external subsystems. If you omit the REO, IMS assumes a default value of R.

Oracle Access Manager for IMS TM also provides a mechanism so that you can specify the REO at the application level on the basis of PSB name. An REO specified at the PSB level overrides whatever REO is given or defaulted for the region.

Recovery Considerations

ESAF allows automated recovery after an outage by providing a two-phase commit interface for synchronizing updates. It also provides a process for resolving partially-committed (in-doubt) transactions when connections are reestablished. The IMS control region resolves in-doubt transactions with participation of all ESAF-defined external resources.

To support this recovery, an Oracle Access Manager for IMS TM instance establishes a connection from the control region to the target Oracle database and executes functions to forcibly commit or rollback pending transactions as directed by IMS. The Oracle Access Manager for IMS TM RTT generation must specify a valid Oracle user id under which these recovery actions are performed. The user id does not have full DBA privileges, but does require several privileges that are part of the Oracle DBA role. You can create a new Oracle user id for this purpose or use an existing user id with the required privileges. The RTT generation must provide authentication specifications allowing the recovery user id to connect to Oracle.

Clarification of Cursor Close Behavior

The Oracle server's normal behavior leaves cursors open across COMMIT and ROLLBACK operations. You can use the precompiler MODE option to request application behavior closer to current ANSI standards, including closing of cursors on commit or rollback. The Oracle Access Manager for IMS TM supports this behavior and IMS-initiated commit (GU or SYNC) and rollback (ROLL or ROLB). If MODE is set to ANSI or one of its variations is not specified at precompile time, then Oracle cursors remain open across IMS-initiated commit or rollback. However, if the application is defined in the RTT AMITRANS macro as OID=IMSID and the beginning of a new transaction causes a change in the user id, then cursors are closed and database session state is lost before the new transaction begins. A change in Oracle user id forces the current Oracle session to be deleted and a new one created. It is the responsibility of the application to be aware of this condition and act accordingly.

This situation cannot arise if the AMITRANS macro specifies PSB, PGM, or a fixed user id string as the Oracle user id. It also does not occur if the AMITRANS macro has OID set to IMSID and a new transaction does not convey a change of user id, which might occur when the new input message comes from the same user, LTERM, or from a BMP.

Handling Oracle Unavailable Situations

Oracle unavailable refers to any situation in which the Oracle Access Manager for IMS TM cannot access the target Oracle server. This situation might result from shutdown or failure of a variety of system components including:

- The target Oracle instance
- Oracle Net for z/OS or a specific protocol adapter
- Other network software (for example, TCP/IP, VTAM)

- Physical network components (routers, for example)
- Remote Oracle Net listener

Oracle Access Manager for IMS TM makes no operational distinction among these situations. It recognizes Oracle is unavailable, perhaps temporarily. There is a difference, however, in whether the condition is detected on the Oracle Access Manager for IMS TM initial connection attempt versus a loss of Oracle access after normal connections are established.

Initial Connection Failure

When IMS starts, if the RTT AMIRT macro CONNECT parameter is set to START, then the control region immediately attempts a connection to the target Oracle server. An Oracle-unavailable condition is detected immediately. On the other hand, if the CONNECT parameter is set to DEFER or defaulted, then the control region waits for the first dependent region to make an Oracle request before attempting its own connection. If the control region is unable to connect to Oracle, then Oracle Access Manager for IMS TM instance is placed in a logical stopped state.

When Oracle Access Manager for IMS TM is in a stopped state after failure of its initial connections, all applications issuing requests for that Oracle Access Manager for IMS TM instance receive a U3049 pseudo-abend and are requeued for later processing. This occurs regardless of the REO option. Even when REO is set to R, an application does not receive an Oracle error when the Oracle Access Manager for IMS TM initial connection attempt fails.

When an Oracle Access Manager for IMS TM instance is placed in the stopped state, it must be started with the IMS command /START SUBSYS after the target Oracle server becomes accessible. This state causes Oracle Access Manager for IMS TM to reinitialize and reattempt a connection to Oracle. If the connection attempt is successful, then queued transactions and new transactions process normally. IMS remembers a subsystem is in stopped state over a warm start. Even if IMS is shutdown and restarted, the /START SUBSYS command is required.

Failure After Initial Connection

If the Oracle Access Manager for IMS TM initial connection attempt from the control region is successful, then a subsequent loss of access is handled without placing the instance in the stopped state. The failure is likely to be detected during execution of an application making Oracle requests. Application behavior in this case is governed by the REO in effect for the transaction, if coded on the AMITRANS macro, or the region from the SSM entry.

If REO is set to R (or taken as a default), then the Oracle error code associated with the lost connection is returned to the application. Applications that use option R must be careful to check SQLCODE or the return code from an OCI call. If the application fails to detect the error and issues another Oracle request, then a loop between IMS and Oracle Access Manager for IMS TM can result. Refer to the IBM IMS documentation for more information.

With REO set to Q, an application is abended with U3044, requeued, and the transaction is placed in PSTOP status. With option A, the application receives a U3983 abend and the input transaction is discarded. These abends might cause IMS to invoke Oracle Access Manager for IMS TM to resolve in-doubt processing in the control region. This process also fails if Oracle has become unavailable and IMS holds the associated recovery tokens to be processed later.

These circumstances do not place the Oracle Access Manager for IMS TM instance in the stopped state. Instead, Oracle Access Manager for IMS TM remains active and attempts to reestablish a connection to Oracle when another application makes an Oracle request. If the connection attempt is successful, then Oracle Access Manager for IMS TM resumes normal operation. If it fails, then the application is processed according to the REO in effect.

Each attempt to reestablish the connection to Oracle results in some message traffic to the system console or master terminal operator (MTO) console. A high frequency of failing attempts might result in an excessive message load. A RECONTM parameter on the AMIRT macro can specify the MAXIMUM frequency of reconnect attempts in elapsed second terms. The default for RECONTM is 60 seconds: Oracle Access Manager for IMS TM does not attempt to reconnect on behalf of an application if fewer than 60 seconds have passed since the last attempt. If RECONTM is 0, then every application making an Oracle request causes an attempt to reconnect.

Oracle Environment Variables

Environmental variables control aspects of Oracle product behavior. Environment variables are distinctly-named parameters, set by a mechanism external to the product. Oracle NLS support relies on environment variables to determine the user’s preferred message language, character set encoding, and other locale-sensitive attributes.

In a non-IMS batch job or TSO session, the Oracle Database for z/OS products read a sequential file containing environment variable settings. However, IMS environments prefer different environment variable settings from one transaction to the next in the same region. Sequential file processing might negatively affect performance. Oracle Access Manager for IMS TM specifies environment variables in the RTT generation.

Environment variables are defined in groups in the RTT. Each group is associated with specific transactions (PSBs), specific Oracle user ids, or with the RTT as a whole by coding the group name on the transaction, session, or main RTT definition macros. When an Oracle software component requests the value for a particular variable, Oracle Access Manager for IMS TM checks environment variable groups for:

- Current transaction definition
- Current session definition
- RTT default

These groups are checked in this order to locate a value. The transaction-level specification of a variable has highest priority, followed by a session-level specification, followed by the RTT default.

It is not necessary to provide environment variable definitions at any level if the normal Oracle Database for z/OS defaults are acceptable. The Oracle Access Manager environment does not use the CONNSTR variable, and all variables whose names begin with CRTL_. If these variables are specified, then they are ignored.

Configuring Oracle Access Manager for IMS TM

Oracle Access Manager for IMS TM must be installed successfully before you can configure it. Refer to the *Oracle Database Installation Guide for IBM z/OS* for more information.

Oracle Access Manager for IMS TM

Configuration Checklist

- ❏ Step 1: Define a z/OS Subsystem Identifier for the Instance
- ❏ Step 2: Choose a Value for the Instance and Generate the LIT
- ❏ Step 3: Create a User Id in the Target Oracle Database Used to Conduct Recovery
- ❏ Step 4: Determine the Oracle User Id, Authentication, and Environment Variable

- ❑ Step 5: Code and Generate the Control Region and Dependent Region RTT
- ❑ Step 6: Add a Control Region and Dependent Region SSM Entry for the Instance
- ❑ Step 7: If a New SSM Member is Created for Any Region, Specify the Member to IMS
- ❑ Step 8: Make the Oracle Access Manager Code and Modules Available to IMS Regions
- ❑ Step 9: Shutdown and Restart IMS

When IMS is restarted, the control region reports the status of Oracle Access Manager for IMS TM initialization. The control region automatically initializes Oracle Access Manager for IMS TM in the dependent regions used. Actual connection to the target Oracle database might occur, depending on options specified in the RTT (for example, one option is to defer connection until a transaction actually issues an Oracle request). Transaction programs prepared for use with Oracle Access Manager for IMS TM can execute.

Configuration Steps

Be sure you have read "Integration with IMS" and "Configuration Overview" earlier in this chapter, before beginning the steps in this section.

Step 1: Define a z/OS Subsystem Identifier for the Instance

Oracle Access Manager for IMS TM requires a z/OS subsystem id. The id is used as part of an internal communication mechanism. Any valid one-character to four-character subsystem id known to z/OS can be used as long as it is not used by another subsystem or another instance of Oracle Access Manager for IMS TM.

An IPL normally is required to add new subsystem identifiers to z/OS. However, if the subsystem name you assign is not formally defined, Oracle Access Manager for IMS TM dynamically creates its own entry on the z/OS subsystem control table (SSCT) chain. It creates the entry by using RTT macro AMIRT DYNSUBS set to YES, which is the default. This entry allows the product to be configured and used without requiring a z/OS IPL. The interface for adding an SSCT entry is not a published z/OS interface. If you prefer, you can specify that Oracle Access Manager for IMS TM is not to dynamically create its own SSCT. This is done by specifying the RTT macro AMIRT DYNSUBS be set to NO. In this case, Oracle Access Manager for IMS TM is not able to run until the subsystem name is added and z/OS is re-IPLed.

Step 2: Choose a Value for the Instance and Generate the LIT

The LIT is a four-character identifier. It is generated using an Assembler language macro (AMILI) and embedded at linkedit time in each IMS transaction program using the associated instance of Oracle Access Manager for IMS TM. The LIT you choose must be unique within the IMS subsystem. It cannot duplicate the LIT of another Oracle Access Manager for IMS TM instance or the LITs associated with other external resources.

The LIT can be identical to the subsystem id. However, an identical LIT can cause problems if you expect to vary the LIT subsystem configuration in the future. For example, if you change the SSM so an existing LIT is associated with a new subsystem, the new LIT might be confused with the old subsystem identifier.

LIT generation is performed by coding and assembling an AMILI macro instruction as follows:

```
[name]AMILI [LIT=lit]
```

where:

Table 12–2 Variable Descriptions for Code Example

Variable	Description
`name`	is the CSECT name to use for this LIT generation. If the name is omitted, then the name AMILI000 is used by default. If a name is specified, then ensure it does not conflict with external names occurring in transaction programs. Because only one Oracle Access Manager for IMS TM LIT can be linked into an application, it does not matter if the same CSECT name is used in LITs for different Oracle Access Manager for IMS TM instances.

Table 12–2 (Cont.) Variable Descriptions for Code Example

Variable	Description

`LIT=lit`	specifies the one-character to four-character LIT. The value can be enclosed in apostrophes. It must conform to IMS ESAF requirements, such as alphanumeric characters. If this parameter is omitted, a LIT value of ORA1 is generated.
	
	Assembly of the LIT requires access to the Oracle Access Manager for IMS TM macros. The resulting object code can be linkedited into a load module library or saved as an object deck. It must be included in the linkedit of IMS transaction programs using the associated instance of Oracle Access Manager for IMS TM. The LIT contains no executable code and does not have addressing mode (AMODE) or residency mode (RMODE) limitations.
	
	A sample LIT generation job:
	`//AMILIT1 JOB (ORA),'ORACLE LIT GEN'`
	`//  EXEC ASMCL,PARM.LKED='LIST,RMODE=ANY'`
	`//ASM.SYSLIB  DD DSN=ORACLE.V10G.MACLIB,`
	`//           DISP=SHR`
	`//ASM.SYSIN    DD *`
	`ORA3LIT  AMILI LIT=ORA3`
	`         END`
	`/*`
	`//LKED.SYSLMOD DD DSN=IMS1.DEV.LIB(ORA3LIT),`
	`//          DISP=SHR`
	`//`

Step 3: Create a User Id in the Target Oracle Database Used to Conduct Recovery

Oracle Access Manager for IMS TM requires a user id in the target Oracle database so recovery sessions can be established when needed by the control region. Although you can use an existing Oracle user id with the appropriate privileges, Oracle Corporation recommends creating a distinct user id dedicated to this purpose. The user id must have or be granted these privileges:

- CREATE SESSION (system privilege)
- FORCE ANY TRANSACTION (system privilege)
- SELECT ON SYS.PENDING_SESSIONS$ (object privilege)
- SELECT ON SYS.PENDING_TRANS$ (object privilege)

No other privileges are required. If you are creating a user id dedicated to Oracle Access Manager for IMS TM recovery purposes, then Oracle Corporation recommends only these privileges be granted to the user id.

The recovery user id does not create data in the target database. Therefore, RESOURCE privileges and user id tablespace defaults and quotas are unimportant. The user id profile is important because the recovery user id must not be subject to Oracle resource limits that could cause an interruption of recovery activity. Such activity includes the profile IDLE_TIME limit because the recovery session can stand idle for a long time.

The authentication method for the recovery user id can be password or EXTERNAL. When creating a new user id, choose an authentication method based on your Oracle and IMS security practices. The recovery user id is coded explicitly in the RTT; it is not derived from an RTT transaction specification. The control region RTT must contain a session authentication entry allowing the recovery id to connect successfully to Oracle. This entry can be the default session entry if the default authentication method applies to the recovery id. If it does not, an explicit session entry for the recovery user id must be included in the RTT.

If you plan to access a particular Oracle database from more than one IMS subsystem, all IMS subsystems can use the same recovery user, assuming the user id's profile allows at least one session per IMS. However, Oracle Corporation recommends you create a distinct recovery user id for each distinct IMS subsystem. This allows better activity tracking in the Oracle server and simplifies problem resolution.

SQL statements for creating a recovery user id are shown in the following example. They are suitable for use in a Server Manager or SQL*Plus session:

```
CONNECT SYS/CHANGE_ON_INSTALL
CREATE USER IMS1RECO IDENTIFIED BY RECO1PW PROFILE NO_LIMIT;
GRANT CREATE SESSION, FORCE ANY TRANSACTION TO IMS1RECO;
GRANT SELECT ON PENDING_SESSIONS$ TO IMS1RECO;
GRANT SELECT ON PENDING_TRANS$ TO IMS1RECO;
```

The example assumes a profile named NO_LIMIT is already defined to Oracle. The GRANT SELECT statements specify unqualified table names because the connection is with user id SYS, which owns the PENDING_TRANS$ and PENDING_SESSIONS$ tables.

Step 4: Determine the Oracle User Id, Authentication, and Environment Variable

Review the transaction programs you plan to run with this Oracle Access Manager for IMS TM instance to choose the method for determining the Oracle user id for each transaction. When you know what Oracle user ids to use, establish the authentication method for their Oracle sessions. If you are creating new Oracle user ids for Oracle Access Manager for IMS TM applications, you can choose whichever authentication method is appropriate for your installation's security practices.

Finally, consider the user and transaction management requirements to determine environment variable settings at the RTT, session, and transaction level.

Step 5: Code and Generate the Control Region and Dependent Region RTT

Four macros are used to code the RTT:

- AMIRT
- AMITRANS
- AMISESS
- AMIENV

For a summary of RTT macro parameters governing session cache, refer to "Clarification of Cursor Close Behavior" on page 12-9.

AMIRT

The AMIRT macro is invoked once or twice in an RTT definition. The first invocation specifies the connection string for the target Oracle database and other options with subsystem-wide or region-wide scope. The connection string for the target Oracle database is meaningful only in the RTT used by the control region. The address can be omitted in a dependent region RTT. If it is specified, then it must be identical to the one specified for the control region. Otherwise, Oracle Access Manager for IMS TM initialization for the dependent region fails.

The AMIRT macro is coded using this syntax:

```
[name] AMIRT [DBADDR='string']
[CONNECT={START|DEFER}]
[RECOID='string']
[DYNSUBS={YES|NO}]
[ENVTAB=envname]
```

```
[END={YES|NO}]
```

where:

Table 12–3 Variable Descriptions for Code Example

Variable	Description
`name`	can be any name allowed by the assembler. It is ignored.

`DBADDR='string'`	specifies the Oracle connection string of the target Oracle database. This string must be a complete address string (not a TNSNAMES alias identifier). Refer to the example in the note at the end of this `DBADDR='string'` description. The address is normally enclosed in apostrophes because it can contain special characters such as blanks and parentheses. This parameter can be omitted in a dependent region RTT and when coding AMIRT END set to YES to conclude an RTT definition.
	
	If accessing a remote database, Oracle Net address strings can be lengthy. You need to code continuations of the line on which DBADDR is specified. Remember the assembler normally requires a nonblank continuation indicator in position 72 and the continuation begins in position 16 of the next record. Positions 1-15 must be blank.
	
	The easiest way to determine what to specify for an Oracle Net address string is to look in an existing TNSNAMES configuration file. Refer to Chapter 8, "Oracle Net" for more information.
	
	Note: The DESCRIPTION keyword is required. For example, for a local database:
	
	`AMIRT DBADDR='(DESCRIPTION=`
	`(ADDRESS=(PROTOCOL=XM)(SID=QA74)))'`
	
	or, for a remote database:
	
	`AMIRT  DBADDR='(DESCRIPTION=(ADDRESS=`
	`(PROTOCOL=TCP) (HOST=144.25.40.217)  (PORT=1521) (SSN=TNS))`
	`(CONNECT_DATA=(SID=QA74)))'`

`CONNECT={START|DEFER}`	specifies whether the region is to establish the connection to Oracle at region startup (START) or wait until the first Oracle access request is made by a transaction program (DEFER). The default is DEFER. This option applies to both control and dependent (MPP) regions. However, the control region is required to establish a connection before any dependent region can connect. CONNECT set to DEFER for the control region serves no purpose if any dependent region immediately uses CONNECT set to START. This parameter is ignored for BMP and IMS fast path (IFP) regions, which always operate with the equivalent of CONNECT set to DEFER.

Table 12–3 (Cont.) Variable Descriptions for Code Example

Variable	Description
`RECOID='string'`	specifies the Oracle user id used for IMS recovery activity. During Oracle Access Manager for IMS TM startup processing, the control region accesses Oracle using this id if there are pending uncommitted transactions. An AMISESS macro specifying this id or a default entry allowing this id to logon to Oracle, must be included in the control region RTT. This parameter is ignored in a dependent region RTT.
`DYNSUBS={YES|NO}`	specifies whether Oracle Access Manager initialization for the control region is permitted to create the Oracle Access Manager subsystem name entry if the name is not defined formally to z/OS. (Refer to the discussion of this topic in "Step 1: Define a z/OS Subsystem Identifier for the Instance" on page 12-13.) The default for this parameter is YES. The parameter is ignored in a dependent region RTT.
`ENVTAB=envname`	specifies the name field of an AMIENV macro coded in the same RTT generation. It must conform to Assembler name syntax requirements. The environment variable specifications in the named AMIENV are used in all Oracle Access Manager processing unless overridden by an AMIENV set specified in an AMITRANS or AMISESS macro in this RTT generation.
`END={YES|NO}`	specifies the end of the RTT definition. When the RTT contains one or more uses of the AMITRANS or AMISESS macros, the AMIRT macro is invoked once at the beginning (where DBADDR is specified) and once at the end with only END set to YES specified. In an RTT definition containing no AMITRANS or AMISESS macros, AMIRT can be invoked a single time with parameters and END set to YES combined.

AMITRANS

The AMITRANS macro assigns characteristics to IMS applications by PSB names. It determines the Oracle user id that causes a transaction's Oracle access. The Oracle user id assignment can be a fixed value (for example, SCOTT) or it can be one of three dynamic values associated with the transaction instance. You can assign different PSB names to different user id determinations. A default determination can be specified for PSB names that do not have an individual entry.

An RTT definition does not need to include AMITRANS macros. The RTT definition assumes a single default user id determination method for all transactions that run in a region without macros. AMITRANS macros used in an RTT definition, must appear after the first AMIRT macro. AMIRT with END set to YES must also appear at the end of the RTT definition.

The characteristics of one or more individual transactions can be specified in a single use of AMITRANS.

The AMITRANS macro is coded as:

```
[name]   AMITRANS  PSB=(psb1,...)
   OID={IMSID|PSB|PGM|'string'}
 [REO=c]
 [ENVTAB=envname]
```

where:

Table 12–4 Variable Descriptions for Code Example

Variable	Description
`name`	can be any name allowed by the assembler. It is ignored.
`PSB=(psb1...)`	specifies the IMS PSB names of the applications to which a common set of characteristics are being assigned. A name specification of * designates a default for transactions for which no specific entry is given in any AMITRANS macro. The * can be included on an AMITRANS call also listing specific PSB names. No more than one * entry can be created in a single RTT definition and no specific PSB name can be repeated in an RTT.
`OID={IMSID|PSB|PGM|string}`	specifies the method of determining the Oracle user id for the transactions listed. The choices are:
`IMSID`	is the z/OS and IMS authorization id (or a substitute) used.
`PSB`	is the IMS PSB name used.
`PGM`	is the program name used.
`string`	is the fixed character string used. This value can be enclosed in apostrophes. It must be a valid user id in the target Oracle database and must conform to the content rules for Oracle user ids.
`REO=c`	specifies the IMS region error option used with the associated transactions. This option is coded as a single letter. Permissible values and their meanings are discussed in Error Processing.
`ENVTAB=envname`	specifies the name of an AMIENV macro coded in the same RTT generation. It must conform to Assembler name syntax requirements. The environment variable specifications in the named AMIENV are used in all Oracle Access Manager processing for the indicated transactions. They take precedence over specifications of the same variable in environment tables associated with the AMISESS and AMIRT macros.

When OID is set to IMSID, determination of the user id varies with the environment. For message-driven transactions:

- Use the RACF-validated signon id if the terminal originating the transaction is signed on.
- Use the LTERM id if the terminal is not signed on

In a non-message-driven region:

- If the address space is present, use the ASXBUSER field.
- If the address space is not present, use the IMS PSB name.

If an RTT contains no AMITRANS macros, the region operates with this default:

```
AMITRANS PSB=*,OID=IMSID
```

This statement specifies that all transactions use the MVS/IMS user id (or a substitute) as the Oracle logon id. If the RTT contains any AMITRANS macros, this default is not assumed. If the RTT contains one or more AMITRANS macros and you also want to specify default transaction characteristics, you must add an AMITRANS for PSB=* or * to the PSB list of an existing AMITRANS macro call in the RTT definition.

AMISESS

The AMISESS macro is used to indicate, by Oracle user id, the type of logon authentication to use and, if required, the Oracle logon password.

An RTT definition does not need to include any AMISESS macros. In this case, a default set of session characteristics is assumed for all sessions created by the region. If any AMISESS macros are used in an RTT definition, they must appear after the first AMIRT macro. A second use of AMIRT with END set to YES is required at the end of the RTT definition.

The session characteristics of one or more Oracle user ids can be specified in a single use of AMISESS.

The AMISESS macro is coded as:

```
[name]   AMISESS  OID=('string' [,'string'...])
    AUTH={EXTERNAL|'string'}
    [ENVTAB=envname]
```

where:

Table 12–5 Variable Descriptions for Code Example

Variable	Description
`name`	can be any name allowed by the assembler. It is ignored.
`OID=('string' [,'string...])`	specifies the Oracle user ids whose session characteristics are being described. A user id specification of * designates a default for user ids for which no specific entry is given. The * can be included on an AMISESS call also listing specific user ids. No more than one * entry can be created in a single RTT definition.
`AUTH={EXTERNAL| 'string'}`	specifies how the user id is authenticated at Oracle logon. EXTERNAL indicates Oracle is to assume the user is already authenticated by IMS or z/OS. In this case, no password is sent to Oracle at logon time. Oracle verifies the user id is known and is created with the IDENTIFIED EXTERNALLY option. If the connection to Oracle is through Oracle Net, the Oracle instance must be configured to allow such connections through Oracle Net. **Note:** If EXTERNAL is specified, it is important to review the LOGON_AUTH setting in the Database Service definition. The alternative to EXTERNAL is to specify the Oracle logon password. The value specified can be enclosed in apostrophes and must match the Oracle user id password. Only one password can be specified per AMISESS macro, so an AMISESS macro with more than one OID value associates the same password with all of the user id. **Note:** RTT passwords are stored in an encrypted form that can be decrypted. It is the user's responsibility to secure the RTT module, for example, by RACF-protecting the RESLIB library in which the RTT is stored.
`ENVTAB=envname`	specifies the name field of an AMIENV macro coded in the same RTT generation. It must conform to Assembler name syntax requirements. The environment variable specifications in the named AMIENV are used in all Oracle Access Manager processing for the indicated sessions unless overridden at the AMITRANS level.

If an RTT contains no AMISESS macros, the region operates with this default:

```
AMISESS OID=*,AUTH=EXTERNAL
```

This default specifies that all transactions connect to an Oracle instance using the external authentication mechanism. Whatever Oracle user id that a transaction uses must be known to the target Oracle instance and have the IDENTIFIED

EXTERNALLY attribute. If the connection uses Oracle Net, the target Oracle instance must be configured to permit externally authenticated logons through Oracle Net.

If the RTT contains any AMISESS macros, this default is not assumed. If the RTT contains one or more AMISESS macros and you also want to specify default session characteristics, you must add an AMISESS for OID=* or * to the OID list of an existing AMISESS macro call in the RTT definition.

AMIENV

The AMIENV macro defines values for various Oracle software environment variables, particularly NLS support. One AMIENV macro call defines a distinct set of variable name and value pairs. The AMIRT, AMITRANS, and AMISESS ENVTAB parameters allow a given set to be associated at the region, transaction, or session level. When a variable is specified at multiple levels, the order of precedence is AMITRANS, AMISESS, AMIRT.

AMIENV allows you to specify any environment variable names. The names and values are not validated during the RTT generation. Do not misspell the name of a variable because AMIENV treats it as though the variable is not specified. An erroneous value for an environment variable becomes apparent at runtime when the software attempts to use the value. How the error is reported depends on the variable and specified value.

In the RTT generation, all AMIENV macros must appear after the first AMIRT macro. They can be mixed with AMITRANS and AMISESS macros and, unlike AMITRANS and AMISESS, also can appear after the END set to YES call to AMIRT.

The AMIENV macro is coded as:

```
name  AMIENV (n1,v1,...)
```

where:

Table 12–6 Variable Descriptions for Code Example

Variable	Description
`name`	is a name field conforming to Assembler rules. The name field is required and uniquely identifies the set of variables within the RTT. It is specified as the ENVTAB parameter of an AMIRT, AMITRANS, or AMISESS macro in the RTT.
`n1`	is an environment variable name, optionally enclosed in apostrophes. Environment variable names used by Oracle products are generally all uppercase.

Table 12–6 (Cont.) Variable Descriptions for Code Example

Variable	Description
`v1`	is the value assigned to the environment variable, optionally enclosed in apostrophes. The apostrophes are required if the value includes characters such as blanks or punctuation that are not part of the Assembler's name syntax. Up to 256 name and value pairs can be specified within the parentheses. A given variable name can appear no more than once in a single AMIENV macro.

Step 6: Add a Control Region and Dependent Region SSM Entry for the Instance

If you access external subsystems from IMS, then you already have an SSM parameter file. If not, you must create one. The parameter file is a member of the IMS PROCLIB data set. Its member name is in the form:

```
imsidSSM_suffix
```

where:

Table 12–7 Variable Descriptions for Code Example

Variable	Description
`imsid`	is the IMS system id.
`SSM_suffix`	is a one-character to four-character suffix you choose. The suffix is passed to IMS using the SSM startup region parameter.

IMS TM Version 4.1 introduces a new keyword syntax for SSM entries. This syntax is not supported by Oracle Access Manager for IMS TM. You must code the SSM entry using IMS/DC Version 3.1 positional syntax, which is accepted by both IMS versions.

An Oracle Access Manager for IMS TM entry in the parameter file such as IMSLORA0, is a single logical record:

```
ssn,lit,ORAESSD,rtt,reo,crc
```

where:

Table 12–8 Variable Descriptions for Code Example

Variable	Description
ssn	is the one-character to four-character subsystem id for this Oracle Access Manager for IMS TM instance. This parameter is required and must begin in the first position of the record.
lit	is the one-character to four-character LIT for this Oracle Access Manager for IMS TM instance. This parameter is required.
`ORAESSD`	is the name of Oracle Access Manager for IMS TM external subsystem module table (ESMT). This parameter is required and must be specified as shown.
rtt	is the load module name for the generated Oracle Access Manager for IMS TM RTT. This module is placed in the IMS RESLIB or in a data set concatenated to RESLIB. This parameter is required in the control region SSM and optional in a dependent region SSM. If omitted in a dependent region SSM, the region uses the RTT specified for the control region.
reo	is the region error option, a one-character value specifying how Oracle Access Manager for IMS TM is to handle external subsystem failures during application request processing. It includes the situation where an application issues a request before a connection to the external subsystem can be made. This parameter is optional. If omitted, the IMS default is R. Oracle Access Manager for IMS TM also allows the REO to be specified on a PSB-by-PSB basis in the RTT. The REO specified at the PSB level overrides this REO. The allowed values are: `R` returns an Oracle error code to the application `Q` abnormally terminates the application with a U3044 abend code. The input transaction is requeued to be processed when an external subsystem connection becomes possible. `A` abnormally terminates the application with a U3983 abend code. The input transaction is discarded.
crc	is an optional command recognition character. This parameter is not used and can be omitted.

Step 7: If a New SSM Member is Created for Any Region, Specify the Member to IMS

If in Step 6 you created a new SSM member, you must specify the member to IMS. Both the control and dependent regions use the SSM keyword parameter for this purpose. The value for the parameter is the one-character to four-character *SSM_suffix* discussed in Step 6.

The SSM parameter can be specified in the execute procedures for the control region or dependent regions and can be specified in the JCL for batch message processing jobs. For IMS Version 4 or higher, it can be included in the operator command for /START SUBSYS. Refer to the IBM document *IMS System Definition and Tailoring* for more information on the coding of the SSM parameter.

Step 8: Make the Oracle Access Manager Code and Modules Available to IMS Regions

Access Manager for IMS uses modules from an ORAAMIDD DD statement, if present in IMS jcl. This is an alternative for placing authorized modules in the STEPLIB concatenation. The generated RTTs must be available in a data set that is part of STEPLIB or ORAAMIDD and DFSESL.

Copy the following modules into an authorized library that is part of each region's STEPLIB or ORAAMIDD and DFSESL concatenation:

- ORAAMSD (external subsystem code, copy from AUTHLOAD)
- ORAESSD (external subsystem table, copy from AUTHLOAD)
- AMIUS (ami messages, from MESG data set)

Note: The Oracle AUTHLOAD cannot be used in the DFSESL concatenation because IMS does not support PDSE format in DFSESL.

Copy the following modules from CMDLOAD into an authorized library that is a part of each region's STEPLIB concatenation or ORAAMIDD:

- LIBCLNTS (shared client code)
- AMIMAIN (IMS client interface code)
- AMIDMYC (bootstrap program)

Note: This library must be in PDSE format and should not be put in the DFSESL concatenation.

Make the Oracle messages available to the region by performing one of the following actions:

- Authorize the `MESG` data set and add it to the STEPLIB concatenation.
- Add an `ORA$LIB` DD statement for the MESG data set.

Step 9: Shutdown and Restart IMS

When appropriate, follow your installation’s normal procedure to shutdown and restart the IMS subsystem. You can use a warm start or cold start.

If your configuration is successful, you see message AMI-0108 displayed at the IMS MTO console once for each region that is permitted to access the new Oracle Access Manager instance. If the RTT for a region specifies CONNECT set to START (on the AMIRT macro), you also see message AMI-0113 indicating a connection is established to the target Oracle instance. Otherwise, IMS delays the connection attempt until a transaction issues a request.

If the connection attempt fails for any reason, an error message is displayed and IMS places the Oracle Access Manager instance in a stopped state. Once you have corrected the problem, you can restart the Oracle Access Manager instance by issuing the IMS /START SUBSYS *`ssn`* command.

For more information on operating the Oracle Access Manager, refer to "Starting and Stopping Oracle Access Manager for IMS TM" on page 12-29.

IMS External Subsystems

IMS refers to Oracle Access Manager for IMS TM as an external subsystem. IMS requires a z/OS subsystem identifier for each instance of Oracle Access Manager for IMS TM you configure. However, the product does not run as a separate z/OS address space. Instead, the Oracle Access Manager code runs inside the IMS control region and in each dependent region that accesses Oracle data. A separate address space is required for an Oracle Database for z/OS database or for Oracle Net for z/OS. Oracle Database for z/OS or Oracle Net for z/OS is used when running Oracle Access Manager for IMS TM. Refer to Chapter 8, "Oracle Net" for Oracle Net for z/OS operating considerations.

The IMS control region is responsible for monitoring the status of all external subsystems, including all configured instances of Oracle Access Manager for IMS TM. It views each instance as being stopped or started. You can query IMS to determine the status of a subsystem using the IMS operator command DISPLAY SUBSYSTEM. Refer to the IBM IMS documentation for a detailed description of the DISPLAY SUBSYSTEM command.

For example, to request the status of a subsystem with the identifier AMI1, issue the IMS command:

```
/DIS SUBSYS AMI1
```

from the IMS master terminal operator (MTO) or z/OS system console. IMS might respond with something like:

```
DFS000I SUBSYS  CRC   REGID   PROGRAM    LTERM     STATUS
DFS000I AMI1    #                                  NOT CONN
```

In this example, the Oracle Access Manager for IMS TM instance `AMI1` is not currently connected to the target database. If the subsystem is started and active in several regions, then the display might look like:

```
DFS000I SUBSYS  CRC    REGID   PROGRAM   LTERM     STATUS
DFS000I AMI1    #          1   PRGL08    HN1LN006  CONN
DFS000I AMI1    #          3   PRGL15Q   HN1LN083  CONN
DFS000I AMI1    #          4   PSVB      HN0LN19A  CONN
DFS000I AMI1    #         12   PRGNQ     HN1LN072  CONN
DFS000I AMI1    #         14   PBNC2UP             CONN
```

Although Oracle Access Manager for IMS TM does not have a display command, it does display status messages under circumstances such as:

- Startup
- Termination
- Recovery activities
- Errors

These messages always begin with the prefix AMI- followed by a unique message number. The complete set of Oracle Access Manager for IMS TM messages are documented in the *Oracle Database Messages Guide for IBM z/OS*.

All Oracle Access Manager for IMS TM messages are written to the operator console and the MTO.

Starting and Stopping Oracle Access Manager for IMS TM

When the IMS control region is started, IMS attempts to initialize all external subsystems configured for that region. This means initializing all external subsystems the IMS subsystem can access. Dependent regions can initialize fewer or even no subsystems, depending on how they are configured.

Initialization of Oracle Access Manager for IMS TM in a given region validates configuration data and allocates virtual memory for required data structures. It does not necessarily attempt to make a connection to the target Oracle server. An Oracle Access Manager for IMS TM configuration option can specify connection to the server be deferred until an IMS application makes a request for Oracle data. This option can be set on a region-by-region basis.

There are connection dependencies. IMS requires the control region establish a connection before any dependent region connection is allowed. If the deferred connection option is used for the control region but a dependent region starting immediately does not defer connection, then the control region is forced to connect immediately.

Regardless of whether connection is made immediately or deferred until the first Oracle request, there are a variety of reasons why the Oracle Access Manager for IMS TM connection process might fail. The target Oracle server might be down. If the target server is remote, then any of a number of required components, such as Oracle Net for z/OS, networking software or hardware, or Oracle Net on the target platform, might be down.

If the connection process fails, then Oracle Access Manager for IMS TM displays error information and informs IMS. IMS places the subsystem in the logical stopped state. If the connection is deferred and is associated with an application request for data, then IMS terminates and requeues the requesting transaction.

When Oracle Access Manager for IMS TM connection fails and the subsystem is stopped, it remains so until the operator issues IMS command START SUBSYSTEM.

Oracle Access Manager for IMS TM does not automatically notify IMS to start when the cause of connection failure is resolved. When the START SUBSYSTEM command is issued, Oracle Access Manager for IMS TM reinitializes in each region. Connection to the target Oracle server is attempted or deferred as indicated in configuration data, just as in the original region startup.

Failures and Recovery

Normal IMS and Oracle Access Manager processing can be interrupted by a failure of any of the hardware and software components involved. Because Oracle Access Manager for IMS TM supports IMS two-phase commit synchronization, active transactions at the time of a failure are one of the following:

- Completely committed
- Completely rolled back

- Left in a recoverable state called in-doubt

When problems are rectified and normal IMS and Oracle Access Manager for IMS TM processing resumes, the IMS control region performs a resolve in-doubt process. This process determines the status of each in-doubt transaction and commits it or rolls it back accordingly. Oracle Access Manager for IMS TM displays messages during this process, indicating the action taken with each in-doubt transaction.

IMS also performs resolve in-doubt processing any time an application abnormally terminates. Therefore, the same messages can be expected after an abend in a transaction using Oracle Access Manager for IMS TM.

From the Oracle server side, transaction activity from Oracle Access Manager for IMS TM falls into the same general class as Oracle distributed transactions. The same internal Oracle tables and views record the status of all such transactions. Transactions that originated in Oracle Access Manager for IMS TM can be identified by their parent database id. Oracle Access Manager for IMS TM constructs a parent database id using a combination of the IMS id and the Oracle Access Manager for IMS TM subsystem id. Detailed information on these tables and views can be found in *Oracle Database Reference.*

Oracle Access Manager for IMS TM Storage Requirements

This section describes the storage requirements for Oracle Access Manager for IMS TM.

Base Code Storage Requirements

Table 12–9 lists the base code storage requirements for Oracle Access Manager for IMS TM.

Table 12–9 Access Manager for IMS TM Base Code Storage Requirements

CSA	Storage
Base requirement	500 Bytes
Control region RTT and Dependent region RTT	Varies according to customer definition
Total	500 Bytes + RTT size

All CSA allocations are ECSA (above 16M).

Adapter Storage Requirements

Table 12–10 lists the control region adapter storage requirements for Oracle Access Manager for IMS TM.

Table 12–10 Access Manager for IMS TM Adapter Storage Requirements, Control Region

Control Region	Storage
ORAAMSD	14K
LIBCLNTS	28M

All control region allocations are extended private (above 16M).

Table 12–11 lists the dependent region adapter storage requirements for Oracle Access Manager for IMS TM.

Table 12–11 Access Manager for IMS TM Adapter Storage Requirements, Dependent Region

Dependent Region	Storage
ORAAMSD	14K
LIBCLNTS	28M
Oracle Client storage	100K

All dependent region allocations are extended private (above 16M).

13 Oracle Enterprise Manager Management Agent

This chapter introduces the Oracle Enterprise Manager Management Agent, and provides configuration information.

The following topics are included:

- Overview
- Configuring the Management Agent
- Controlling the Management Agent

Overview

The Oracle Management Agent runs jobs and events sent by the Oracle Enterprise Manager and can start up and shut down a database. It can function regardless of the status of the network connection, and it can run even if the database is down. It enables the database administrator to monitor and control a database from a central location on the network.

The Management Agent is controlled using a command in the z/OS UNIX System Services shell environment. Currently, it is the only component of Oracle Enterprise Manager that is implemented for Oracle Database for z/OS. For more information on the Management Agent, refer to the *Oracle Enterprise Manager Book Set.*

Configuring the Management Agent

The Management Agent is installed with Oracle Database for z/OS. Oracle Universal Installer also configures most of the required parameters.

For the Management Agent to work correctly on z/OS, the following changes must be made to the system:

- The Mangement Agent is a multi-threaded application which starts and stops numerous threads during its operation. The BPX parameter file needs to be altered and activated on your system, as follows:

  ```
  MAXTHREADS(20000)
  ```

  BPX parameters are set in the SYS1.PARMLIB(BPX*xxx*).

- The nmo and nmp executable needs to have its extended attributes set as follows:

  ```
  extattr +p nmo
  extattr +p nmb
  ```

  This is because the executable uses the c function `__passwd` to verify a user and password before attempting to execute a command on behalf of a user (for example, database shutdown and startup).

- The ownership of the `em*` executables in the `ORACLE_HOME/bin` directory must be set to that of the user who will run the Management Agent.

To complete the configuration, the following three files need additional customization:

- The `emd.properties` file in the `sysman/config` directory

  This file contains the global variables for the Management Agent. For most installations, the following two parameters need to be modified:

  – `REPOSITORY_URL=http://host:port/em/upload/`

    This is the location of the Oracle Management Service where you want the Management Agent to register.

  – `EMD_URL=http://local_host:port/emd/main/`

    This is the location and port where you want the Management Agent to listen for communications from clients.

- The `targets.xml` file in the `sysman/emd` directory

  This file contains the information about the local system and its database instances. It is in XML format and needs to be customized. Although the Management Agent attempts to configure this file, z/OS differences may not permit it to do so.

The following is a sample `targets.xml` file:

```
<?xml version = "1.0"?>
<Targets AGENT_SEED="153361320">
 <Target TYPE="oracle_emd" NAME="local_host:port" VERSION="1.0"
 <Target TYPE="host" NAME="local_host" VERSION="1.0"/>
 <Target TYPE="oracle_database" NAME="DB_sid" VERSION="1.0">
  <Property NAME="MachineName" VALUE="local_host"/>
  <Property NAME="Port" VALUE="DB_port"/>
  <Property NAME="SID" VALUE="DB_sid"/>
  <Property NAME="ConnectDescriptor"
VALUE="(DESCRIPTION=(ADDRESS_LIST=(ADDRESS=(PROTOCOL=TCP)(local_host)(PORT=DB_port)))(CONNECT_DA
TA=(SID=DB_sid)))"/>
  <Property NAME="OracleHome" VALUE="oracle_home"/>
  <Property NAME="UserName" VALUE="DBSNMP"/>
  <Property NAME="password" VALUE="password" ENCRYPTED="FALSE"/>
  <Property NAME="Role" VALUE="NORMAL"/>
 </Target>
</Targets>
```

Variables for the previous example are defined as follows:

Table 13–1 Variable Descriptions for Code Example

Variable	Description
AGENT_SEED=	The value of this parameter should be the same as that of the `agentseed` parameter in the `emd.properties` file
`local_host`	Host of the EMD_URL
`port`	Port number of the EMD_URL
`oracle_database`	Oracle instance on the local system or a remote system

The following variables for the previous example can be obtained from the `tnsnames.ora` file for the instance being monitored. They are defined as follows:

Table 13–2 Variable Descriptions for Code Example

Variable	Description
`local_host`	Local host name of the system (for example, mvs05.us.oracle.com)
`DB_port`	Port on which the instance is listening

Table 13–2 (Cont.) Variable Descriptions for Code Example

Variable	Description
`DB_sid`	Instance ID of the Oracle instance. Corresponds to the `dbname` parameter in the `init.ora` file
`oracle_home`	Location of the Oracle database software
`UserName`	DBSNMP database user account
`password`	DBSNMP database user account password Once the Management Agent starts up, it alters the `targets.xml` file to encrypt this password.

- The `/var/opt/oratab` file

  This file should contain one line for every database instance being configured, as in the following example:

  ```
  instance_name:ORACLE_HOME:N
  ```

  In the previous example, *`instance_name`* is the name of the database instance, and *`ORACLE_HOME`* is the Oracle home where the Oracle database software is located.

Controlling the Management Agent

You use the `emctl` command to control the Management Agent. For a description of this and other Management Agent commands, refer to the *Oracle Enterprise Manager Advanced Configuration* manual.

To issue this command, first log in to the z/OS UNIX System Services environment as the user who owns the Management Agent code. This user also needs DBA privileges.

Set up environment variables in the ENV file located in the Oracle home directory.

To start the Management Agent, use the following command:

```
$ emctl start agent
```

To query the status of agent, use the following command:

```
$ emctl status agent
```

To upload the information of Management Agent to the Management Service, use the following command:

```
$ emctl upload
```

The host information appears at the Management Service location. you need to upload the information only if the status of the agent is not automatically updated.

To stop the Management Agent, use the following command:

```
$ emctl stop agent
```

The log files and tracing information will be written to the `ORACLE_HOME/sysman/log` file.

14

Oracle Real Application Clusters

This chapter provides guidelines for configuring Oracle Real Application Clusters after installing Oracle Database for z/OS. The following topics are included:

Overview

With the Oracle Real Application Clusters option, separate Oracle Database for z/OS instances run simultaneously on one or more nodes (z/OS images) and share a single physical database.

It is important to note that the z/OS implementation follows the Oracle Real Application Clusters design guidelines. The only difference is the use of z/OS filetypes. For more information, refer to the *Oracle Real Application Clusters Book Set*.

Oracle Real Application Clusters has the following characteristics:

- One or more instances of Oracle Real Application Clusters can be started on a given node in the parallel sysplex.
- Each instance has a separate SGA and set of background processes.
- All instances share the same data files and control files of a given database.
- All instances can execute transactions concurrently against the same database and each instance can have multiple users executing transactions.
- Row level locking is preserved across the instances.

Applications accessing the database can run on the same nodes as multiple instances of Oracle Real Application Clusters, or on separate nodes of a parallel sysplex, or on distributed systems using the client/server architecture. Oracle Real Application Clusters can be part of a distributed database system. Distributed transactions access the data in a remote database in the same way, regardless of whether the data files are owned by an Oracle Database for z/OS server (in exclusive mode) or an Oracle Real Application Clusters instance (in exclusive or parallel mode).

Other non-Oracle workloads can run on each node of the sysplex or you can dedicate the entire sysplex or part of the sysplex to the Oracle server. For example, an Oracle Real Application Clusters instance and its applications might occupy three nodes of a five-node sysplex, while the other two nodes are used for non-Oracle applications.

Refer to the *Oracle Real Application Clusters Book Set* for more information about the Oracle Real Application Clusters environment. Refer to the concepts in the *Oracle Real Application Clusters Book Set*, especially the discussions on compatibility issues, restrictions, and database design guidelines.

Setting Up Real Application Clusters

The following steps describe important tasks you must perform when setting up an Oracle Real Application Clusters system, but they do not include all the tasks you might need to do for your site. Before performing these steps, review the concepts manual in the *Oracle Real Application Clusters Book Set* for more information.

Checklist for Setting Up Oracle Real Application Clusters:

1. Review and set options in the CREATE DATABASE statement
2. Set up additional threads of redo log files and rollback segments
3. Share all non-VSAM data sets
4. Set up the common OSDI parameters
5. Set up the common Oracle Real Application Clusters initialization parameters
6. Set up the instance-specific Oracle Real Application Clusters initialization parameters
7. Create the startup JCL

Step 1: Review and Set Options in the CREATE DATABASE Statement

Review and set the following options to allow multiple instances of Oracle Real Application Clusters to function properly. For a complete description of the CREATE DATABASE and ALTER DATABASE statements and options, refer to the *Oracle Database Administrator's Guide* and the *Oracle Database SQL Reference*.

MAXDATAFILES   The MAXDATAFILES option of CREATE DATABASE determines the number of data files a database can have. With Oracle Real Application Clusters, databases tend to have more data files and log files than an exclusive mounted database.

MAXINSTANCES   The MAXINSTANCES option of CREATE DATABASE limits the number of instances that can access a database concurrently. The default value for this option under z/OS is 15. Set MAXINSTANCES to a value greater than the maximum number of instances you expect to run concurrently.

MAXLOGFILE and MAXLOGMEMBERS   The MAXLOGFILES option of CREATE DATABASE specifies the maximum number of redo log groups that can be created for the database. The MAXLOGMEMBERS option specifies the maximum number of members or number of copies per group. Set MAXLOGFILES to the maximum number of instances you plan to run concurrently multiplied by the maximum anticipated number of groups per thread.

MAXLOGHISTORY   The MAXLOGHISTORY option of CREATE DATABASE specifies the maximum number of redo log files that can be recorded in the log history of the control file. The log history is used for automatic media recovery of Oracle Real Application Clusters.

For Oracle Real Application Clusters, set MAXLOGHISTORY to a large value, such as 100. The control file can then store information about this number of redo log files. When the log history exceeds this limit, the Oracle server overwrites the oldest entries in the log history. The default for MAXLOGHISTORY is 0 (zero), which disables log history.

Step 2: Set Up Additional Threads of Redo Log Files and Rollback Segments

Each Oracle Real Application Clusters instance requires its own redo log threads and rollback segments. Refer to the concepts manual of the *Oracle Real Application*

Clusters Book Set for how to plan, create, and assign threads of redo and rollback segments to each instance.

Step 3: Share All Non-VSAM Data Sets

Because non-VSAM data sets will be used by multiple instances, ensure that the data sets used in the startup JCL of Oracle Real Application Clusters instances are specified with a DISP=SHR JCL parameter.

Step 4: Set Up the Database Region Parameters

Review and modify the OSDI database region parameters that apply to individual Oracle Database for z/OS instances in one of the members of PARMLIB (for example, `sid`PARM, where `sid` is the database service identifier). OSDI database region parameters are described in "Database Region Parameters" in Chapter 3, "Configuring a Database Service and Creating a New Database". These parameters are required to start individual Oracle Database for z/OS instances. To enable Oracle Real Application Clusters, set the CLUSTER_ENABLE parameter to YES.

Step 5: Set Up the Common Oracle Real Application Clusters Initialization Parameters

Review and modify the Oracle Real Application Clusters initialization parameters that apply to all instances in the `sid`INIT member of PARMLIB. The `sid`INIT member might also contain other parameters which are not directly applicable to Oracle Real Application Clusters but apply to all instances. For a complete list of parameters common or identical across all instances, refer to the *Oracle Real Application Clusters Book Set*.

In the `init.ora` parameter file, `_cluster_library=skgxn` must be set.

Step 6: Set Up the Instance-specific Oracle Real Application Clusters Initialization Parameters

Create the Oracle Real Application Clusters initialization parameters that must be unique for each instance in a new member of PARMLIB. You may create as many of these members for as many instances as you plan to run mounting the same database. Each of these members also should specify the common initialization parameter file using the IFILE parameter.

Alternately all parameters can be specified in one member of PARMLIB. For syntax and notation required for using one initialization parameter file, please refer to the *Oracle Real Application Clusters Book Set*.

At a minimum, the INSTANCE_NUMBER, INSTANCE_NAME, and THREAD parameters are recommended for each instance.

Step 7: Operating an OSDI Database Service

Each instance of Oracle Real Application Clusters executes in a manner similar to a single OSDI-based database service. For more information, refer to Chapter 3, "Configuring a Database Service and Creating a New Database" and Chapter 5, "Operating a Database Service".

Before starting or stopping Oracle Real Application Clusters instances or doing backup and recovery operations, refer to the information about these topics in the *Oracle Real Application Clusters Book Set*.

Cross System Communication Facility (XCF)

Oracle Real Application Clusters requires the Cross System Communication Facility, commonly known as XCF. For more information about configuring XCF, refer to the IBM document *MVS Setting Up a Sysplex*.

Resource Name Usage on z/OS

Oracle Real Application Clusters instances will, by default, make use of the following z/OS resource names. If these names conflict with other software on your system and this cannot be changed, contact Oracle Worldwide Support for assistance.

XCF Group Names

Each instance will create and join XCF groups with names of the format O*nnnxxxx*, where *nnn* is an internally generated three-character alphanumeric string to uniquely identify a cluster, and *xxxx* is a four-character alphanumeric string to identify the type of XCF group. The following is an example using XCF names for the first Oracle Real Application Clusters instance:

```
XCF group for IPC Messages: O001ORAM
XCF group for Node Monitor: O001ORAN
Oracle Groups: O001Gmmm
```

In the above example, the `mmm` are Oracle group numbers and are assigned serially.

ENQ Names

Each instance will also take out global (sysplex-wide) enqueues on resources with names in the format:

```
Qname : OnnnORAG
```

Qname is the queue name to be used for ENQs used by this cluster.

```
Rname : OnnnGxxx#yyy….yyy
```

Rname is the resource name used by ENQs consisting of `xxx` (a three-character alphanumeric string) and `yyy…yyy` (a variable length alphanumeric string, up to 18 characters).

For example, for the first Oracle Real Application Clusters instance with the database name ORACLE10, the following name will be used:

```
O0001ORAG.O001G001#DBORACLE10
```

15 Oracle Database Performance

The performance of your Oracle server depends on a number of factors. This chapter describes some of those factors and provides recommendations for improving and tuning your system to promote better Oracle performance.

The following topics are included:

- Real Storage Requirements
- Oracle Server Storage Requirements
- Tuning on z/OS

Real Storage Requirements

An excellent method for reducing the real storage requirements for Oracle is to place reentrant modules in z/OS link pack areas. Most Oracle modules are link-edited with AMODE set to 31 and RMODE set to ANY. Place such modules in the extended pageable link pack area (EPLPA) above the 16M line. Some modules are linked with RMODE set to 24. Place them in the pageable link pack area (PLPA) below the 16M line. Current versions of z/OS automatically load modules from the link pack area libraries into the appropriate link pack areas.

Keep in mind that under the current z/OS implementation, placing modules into LPA reduces the private area of every address space in the system. In other words, there is a trade-off between code sharing and virtual memory availability.

Note: If you choose to take advantage of this z/OS feature, be aware that any STEPLIB/JOBLIB definition in the JCL has precedence over the placed modules of any link pack areas. Make sure that such modules do not exist in the STEPLIB/JOBLIB libraries so that the link pack area copy will be used instead.

LPA Considerations for Database and Net Regions

Only the subsystem code module (ORASSI) is automatically shared, and it is shared by all Oracle subsystems and services.

The Oracle database and Net regions run different programs from the Oracle AUTHLOAD library. Each Oracle database address space has its own copy of ORARASC and a few other modules, and the Net address space has ORANET and several others modules as well. There is no sharing of this code, even between address spaces of the same database service. Due to an operating system restriction, you cannot put ORARASC into LPA -- doing so makes it impossible to run any other copy of ORARASC (for example, an ORARASC at a different maintenance level), whether from LPA or not. ORARASC is also quite small in size, so it is not necessary for it to be shared. The Net modules, such as ORANET, are also quite small in size. In addition, because a single Net service can be used to access multiple database services, typically only one Net service will be deployed. As a result, sharing of Net code is also typically unnecessary.

If you are running multiple Oracle database regions (from either the same or different instances), an excellent candidate for LPA usage is the Oracle kernel (in other words, the ORACLE module), because it is quite large in size. As described above, prior consideration should be given to the impact on any non Oracle workloads that may be constrained by virtual storage.

LPA Considerations for Local Oracle Users

If your installation will run multiple concurrent local users, you can place the following modules from the Oracle CMDLOAD into the link pack:

- LIBCLNTS

  LIBCLNTS contains the interface routines for all Oracle client accesses. This includes tools, utilities, precompilers, precompiler applications and Access Managers. LIBCLNTS is an excellent candidate because of it's relatively large size. Place it in the EPLPA above the 16M line.

- SQLPLUS

SQLPLUS is the primary Oracle batch and interactive SQL processor. It is used for user database queries, updates, table creates and drops, and so forth. In many systems, this module is used heavily enough to warrant link pack area placement. Place it in the ELPA.

Oracle Server Storage Requirements

The Oracle server makes static-fixed, static-variable, and dynamic virtual memory allocations as the Oracle regions are started up and begin providing database services to users. Static-fixed memory allocations are storage areas that are always allocated in the regions including space for the Oracle load modules, working storage, and z/OS data areas. Static-variable memory -- mainly the SGA -- differ from one warm start of the server to the next, depending on initialization parameter values. Dynamic memory allocations occur as users connect to the instance and access information that is stored in the server. The primary factors determining the number of concurrent Oracle users that can be supported under z/OS are the user memory allocation requirements (depending on the application design), the INITORA and OSDI parameter values, the amount of virtual memory that Oracle regions are allowed to allocate, and the amount of central storage that is available for use by the Oracle regions.

Database Server Address Space Configuration

Exhausting virtual memory in an address space will lead to any of a number of types of failures, because it is impossible to predict which system activity requests for memory are going to be denied.

This scenario is best avoided by configuring a database instance with enough address spaces to contain the largest expected workload in terms of memory required. Doing this requires an understanding of the workload as well as of the database address space topography on z/OS.

In addition to carefully configuring server address spaces, you can use certain database region parameters that provide controls designed to reduce the likelihood of exhausting address space memory. These are discussed in "Limiting Sessions in a Server Address Space" on page 15-8, and "Limiting Memory Allocations in a Server Address Space" on page 15-8.

Determining the Number of Oracle Address Spaces

Each database address space starts out with a given amount of private virtual memory: 2048 megabytes less the memory that is used or reserved by z/OS for shared access by all address spaces: SQA, CSA, LPA, and the z/OS nucleus and related data. The sizes of these spaces, and thus the amount of private memory remaining in each address space, varies from one z/OS system to another. You may need to consult with your systems staff to determine the available private area size on your system.

Once you know the private area size of your system, you must subtract from it the amount of memory that will be allocated in each address space for purposes other than Oracle sessions. This memory allocation includes the SGA, the Oracle kernel -- usually named ORACLE -- and other minor load modules (ORARASC, ORARSSRB, ORADIE), and the load modules and data structures of the z/OS Oracle infrastructure, including the IBM Language Environment (LE) interface. The size of the SGA is determined primarily by parameters that you specify in the INITORA file and is displayed during Oracle startup. (For additional discussion of the SGA, refer to section "Oracle SGA on z/OS" on page 15-5.) The size of the Oracle kernel and other modules can be determined using ISPF browse on the load library that contains it.

After the foregoing are subtracted, the remaining private memory in each server address space is available for Oracle sessions. The maximum amount of memory that is required by a given session depends mainly on the behavior of the application: the number of cursors opened, the specific SQL statements used, PL/SQL and/or Java requirements (if any), and so forth. In addition, there are several INITORA parameters (SORT_AREA_SIZE, HASH_AREA_SIZE, and so forth) and a database region parameter (INIT_STACK_SIZE) that affect the memory resources that are allocated during each session execution. This can be quite difficult to estimate in advance of running the application. The most reliable way to determine memory requirements is to review the Oracle session SMF records (which contain a session memory high-water mark) and analyze them to determine the average peak session memory. For more information on Oracle SMF records refer to "Interpreting an Oracle Accounting SMF Record".

Once you know the average session memory requirement you can calculate the number of sessions that will fit in one address space as:

```
N = P / S
```

where:

Table 15–1 Variable Descriptions for Code Example

Variable	Description
N	is the desired result

Table 15–1 (Cont.) Variable Descriptions for Code Example

Variable	Description
`P`	is the available private memory per address space
`S`	is the average peak session memory

If we let **T** be the total number of concurrent Oracle sessions to be supported then you need (**T / N**) server address spaces. This number should be rounded to the next higher whole number, and to allow for reasonable variability in workload level, it may be advisable to add one more address space. In doing so, keep in mind that an SQA cost is associated with starting additional address spaces, which is discussed in the section "Oracle SGA on z/OS" on page 15-5.

Oracle Corporation recommends that you specify the number of address spaces calculated here as the INIT_ADR_SPACES parameter (so the address spaces all start when the service is started). A somewhat higher number can be specified as MAXAS (maximum address spaces) on the DEFINE SERVICE command. This makes it possible to start additional address spaces dynamically if the initial estimate proves to be low. There is no cost for having additional address spaces in the MAXAS parameter until those address spaces are actually started. Note that MAXAS **must** be equal to or greater than INIT_ADR_SPACES. Care should be taken to specify a high enough value to accommodate unpredictable workload growth or spikes.

Note that using more than one address space results in the Oracle server becoming a "cross-memory address space" in z/OS terms. These address spaces are not available for reuse when the Oracle server terminates. The z/OS PARMLIB parameter RSVNONR specifies the number of address space numbers to reserve for use as they become unavailable. If you use multiple address spaces for the Oracle server, then you should increase the value specified for this parameter. Specifying too small a value, or letting RSVNONR take the default value could result in an unscheduled IPL if the number of address space IDs becomes exhausted. For example, you could stop the Oracle server for maintenance and then become unable to restart without an IPL. For more information, refer to the IBM document *MVS Initialization and Tuning Reference.*

Oracle SGA on z/OS

An Oracle Database instance has a single SGA, System Global Area, regardless of the number of address spaces or regions configured. The SGA is shared across all of regions of a server using a z/OS service called IARVSERV that allows one address space to "view" a range of private virtual memory that belongs to another address space. The SGA belongs to the first server address space (primary region) and is viewed (shared) by any other regions that are configured for that server. The virtual address range of the SGA must be reserved in each of the auxiliary regions to support the viewing mechanism. This is why the SGA size is subtracted from the private area size in every server address space of a given instance (not just the primary region) in the memory calculations of the previous section, "Determining the Number of Oracle Address Spaces".

Sizing the SGA and, most significantly, sizing the database buffer cache and the shared pool, are important instance tuning activities. The numerous INITORA parameters that do this and the general considerations for specifying their appropriate values are covered in the *Oracle Database Reference* and *Oracle Database Performance Tuning*. The following paragraphs describe some z/OS-specific issues to be aware of when tuning the SGA.

Because the SGA is not permanently pagefixed on z/OS as it is on some other systems, there is little benefit in reserving SGA expansion space with the SGA_MAX_SIZE parameter. When you specify SGA_MAX_SIZE, the indicated maximum size is reserved (in virtual memory) in all server address spaces even if it is not all used.

Keep in mind that the SGA is mapped in all of the server address spaces of a given instance as discussed above. This means that increasing a server's SGA size reduces the virtual memory available for Oracle sessions in every server region for that instance. If you do this, you may need to increase the number of regions in order to support your peak workload. The relationship is **not** linear. A 25% increase in SGA size may require more than a 25% increase in the number of server regions. When you make a significant change in the SGA size, repeat the calculations described in the previous section to determine the number of server address spaces that you need.

Another factor in SGA sizing is the overhead of the IARVSERV memory sharing mechanism. Currently, z/OS must reserve 32 bytes of ESQA (Extended System Queue Area) for each "view" of each 4K page of memory shared. SQA is an expensive resource because it is page-fixed (always backed by real memory) and because it is globally addressable, using up an address range that would otherwise be part of the private area of each address space. Exhausting z/OS SQA is a situation best avoided, so you should calculate the SQA overhead for your SGA and discuss this with your z/OS systems staff before attempting to start the server.

Note: The total amount of SQA to reserve for all uses is a z/OS system initialization parameter and cannot be changed without an IPL.

As an example, an Oracle server configured to run in 10 address spaces with a 512 megabyte SGA requires

32 x 10 x ((512 x 1024 x 1024) / 4096) bytes
or 40 Megabytes, of SQA, a significant amount.

Note: The IARVSERV SQA overhead occurs only when running Oracle servers in two or more address spaces. When a server is configured to run in a single address space only, IARVSERV is not used, and no SQA requirement is imposed. The current IARVSERV implementation provides page-level (4K unit) sharing granularity with a rather high cost in real memory overhead (on the order of 3% of all aggregated virtual views) for mapping tables.

User Stack Area in z/OS

Each Oracle server user requires some extent of private memory to be used as a save area during normal execution. This area is known as the user stack.

When a user session is initiated, the connection is routed to a particular Oracle region. This region will then acquire a stack area based on the INIT_STACK_SIZE parameter. If the user requires more stack, additional extents are dynamically allocated and freed when they are no longer required. The actual stack requirement is dependent on the type of database call being used (SQL, PL/SQL, Java, OCI) and its complexity.

To fine tune this parameter, you might want to use the Oracle session SMF records analysis method described in the "Database Server Address Space Configuration" section (that was presented earlier in this chapter), or you can run your workload by varying the INIT_STACK_SIZE settings and then comparing the CPU usage of the various tests. The lower the INIT_STACK_SIZE value, the higher the potential CPU overhead that is caused by dynamic stack expansion.

As a starting point, you can use the Oracle Corporation recommended minimum value of 128K.

Limiting Sessions in a Server Address Space

The MAX_SESSIONS parameter also plays a role in managing virtual memory use. This is a hard limit on the number of sessions that can be active in one server address space, and it defaults to 1024 sessions. If a new bind (client connection) is routed to an address space that is at the MAX_SESSIONS limit, the server waits until some existing session unbinds (disconnects) before accepting the new session.

The idea behind MAX_SESSIONS is to keep the address space from accepting so many sessions that virtual memory is exhausted and unpredictable failures occur. The assumption is that it is better not to let an application connect and get started than to let it connect and incur transaction failure partway through its processing. A good value to use for MAX_SESSIONS is the value N (maximum sessions per address space) that was calculated in the section "Database Server Address Space Configuration" on page 15-3.

Both INITORA and database region parameter values must be set high enough to allow the required number of users to connect to the server.

Limiting Memory Allocations in a Server Address Space

Two more database region parameters provide additional control over memory consumption in a server address space. The MAX_SESSION_MEM parameter allows you to impose a limit on the total virtual memory allocated to any single session in the Oracle server. This applies to all session-private memory requests made by the server, including the C stack and "heap" areas.

The limit is imposed on all sessions, including background processes and even parallel query slaves. If a session requests memory that would take it over the limit, the session receives an error (usually an ORA-04030) and the current transaction is rolled back.

Care should be taken not to choose too small a session limit. STARTUP processing in the current Oracle release requires about 10 megabytes of session memory. Note that session memory usage is reported in the Oracle SMF record, which can be used to help determine an appropriate limit amount.

Another parameter, REGION_MEM_RESERVE, allows you to limit the total memory allocated to all sessions and the SGA in an address space before exhausting address space private area. The "reserve" amount you specify remains available for internal implementation and z/OS system function use.

Specifying an adequate reserve amount prevents the situation of exhausting address space memory and significantly reduces the impact of memory consumption problems. Requests for memory that would exceed the aggregate limit are rejected, resulting in an error and transaction rollback in the affected session.

Real Storage: Working Set

The amount of real storage ("working set size") that is required by the Oracle regions is very workload dependent, varying significantly with transaction complexity and rate, user concurrency, and locality of program and data reference. A lightly-loaded instance might require only 50K of working set for every megabyte of virtual memory that is allocated, while an instance that is supporting a much higher workload might require 750K of working set for every megabyte of virtual memory that is allocated. In general, Oracle load modules and the SGA remain in central storage while the context areas, sort work areas, and other individual session-related areas are more likely to be paged out to expanded or auxiliary storage when they are not heavily used.

Virtual Memory Allocation

The amount of virtual memory that is consumed by the server regions may vary significantly at run time because memory usage levels are very dynamic and fluctuate according to user workload. This is especially true when users connect and/or disconnect frequently or when users execute applications that open and close a large number of cursors, and when sorts are performed. Region size limitations can therefore become important, even with a small number of users connected to the instance. A region size limit that is too small prevents users from connecting to the server or from accessing information. Oracle Corporation recommends that you allow the server to allocate as much virtual memory as required and that you avoid imposing any region size limitations on the software. Other z/OS facilities can be used to control the amount of central and expanded storage that is used by the server. The easiest way to allow Oracle Database for z/OS to use the maximum amount of virtual storage is to specify the REGION=0M keyword parameter on the EXEC statement in the region startup JCL. For information on the REGION=0M keyword and IEFUSI Exit, refer to the section "Database Region JCL".

Tuning on z/OS

The Workload Manager (WLM) is a z/OS facility that allows installations to effectively manage their Oracle as well as non-Oracle workloads based on business priorities. Goals can be defined to reflect business priorities. The system manages the amount of resources, such as CPU and storage that are necessary for a given workload, to achieve its goal.

Earlier versions of z/OS had a Workload Manager mode called "compatibility mode" where the installation had direct control over performance parameters such as the dispatching priority of the various Oracle address spaces. z/OS 1.4, required for Oracle Database 10*g*, only supports Workload Manger goal mode. With this mode, the installation has no direct control over the detailed performnance parameters, but assigns an importance and a goal to each element of work.

Like other WLM managed workloads, Oracle workloads should be assigned to appropriate service classes based on attributes such as subsystem name, service name, user name, and transaction name. Service class structure and importance are determined by the business needs of an installation. Workloads should also be classified into report classes to facilitate monitoring and validation of an installation's workload management policies.

Oracle Regions

Service classes allow you to control the priority of your Oracle instances relative to other workloads. Service classes should be defined for your Oracle instances based on the performance requirements of the instances. You usually do not need to define a service class for each Oracle instance, because multiple Oracle instances with similar performance requirements can typically be mapped to a given service class (production instances versus test and development instances, for example). Report classes provide more granular reporting capability for different Oracle instances within a given service class and should be used where necessary to monitor CPU, memory, and I/O resources that are consumed by individual Oracle instances in the reports that are generated by SMF/RMF and other measurement subsystems.

Dispatching Priority

The service class of the Oracle regions determines the relative dispatching priority of the background processes and other special tasks within the Oracle instances. The Oracle regions typically consume very moderate amounts of CPU resources. Normally, the bulk of the CPU resources that are consumed to process database requests are incurred by the client address spaces for local requests (and Oracle Net SRB enclaves in the case of remote requests) and should be managed accordingly (refer to "Local Clients" on page 15-12 and "Remote Clients" on page 15-14). Oracle, therefore, typically does not need to run at a high priority, but you may want to consider the special conditions that are associated with the parallel execution feature that is discussed below in the last paragraph of this section.

In general, the Oracle regions can be configured for lower dispatching priority (or lower importance) than high priority CICS TS and TSO workloads, and at about the same priority as high importance (or non-discretionary) batch workload. For example, CICS TS users for a given Oracle instance should be configured for higher priority (or importance) than the corresponding regions of the Oracle instance. Similarly, in the case of TSO, higher priority (or importance) should be assigned to first or second period TSO workloads than to the Oracle regions. A higher dispatching priority is obtained by assigning these address spaces a service class with a high importance. Importance 1 is the highest importance level and Importance 5 is the lowest.

If the Oracle dispatching priority is too low, and if the system suffers from significant CPU contention (indicated by high processor delay in the Oracle regions), then some important Oracle internal requests might not get immediately processed, or the background tasks might not get dispatched often enough to perform the required work. For example, the buffer pool might become filled with modified buffers, and users might need to wait for Oracle to get dispatched and write out some database blocks to allow user processing to continue. The following scenario illustrates this situation:

You see the `'free buffer waits'` event (from a UTLBSTAT/UTLESTAT, STATSPACK, or similar report) showing a significant total value (the unit is hundredths of second) relative to the report interval during a DML-intensive period (update, delete, or insert operations). First, consider enlarging the buffer pool, or pools, to trade memory for I/O requests (you can have multiple buffer pools in Oracle Database for z/OS). Assuming that no significant I/O bottlenecks are affecting the database files (fix them first if any occur), and if the AS1 or Control Address Space CPU delay is low, then it might be necessary to increase DB_WRITER_PROCESSES to schedule more parallel I/O. If the AS1 CPU delay is significant, however, you probably need to set the dispatching priority to a higher value first. Be aware that checkpoint activity also forces modified buffers to disk, adding to the I/O stress. You should also compare the `'physical writes'` and `'physical writes non checkpoint'` statistics from your report to make sure that the write activity is not being unnecessarily inflated by a poorly tuned checkpoint mechanism. Refer *Oracle Database Performance Tuning* for information on minimizing checkpoint overhead and for information on the UTLBSTAT/UTLESTAT and STATSPACK script utilities.

On the other hand, whenever users take advantage of the parallel execution feature that runs under special subtasks in the Oracle regions, the dispatching priority of the Oracle regions becomes an increasingly important tuning issue. In this case, Oracle dispatching priority determines how quickly these special requests are serviced and how much those Oracle users impact the overall throughput of the z/OS system.

Local Clients

TSO

The following considerations apply to resource intensive Oracle workloads within a TSO environment:

1. Increase the relative importance levels of TSO first and second periods. This supports transactions requiring greater resources and may result in a larger percentage of all transactions being completed in the first and second periods.
2. Consider adding a fourth or fifth performance period to account for extremely resource intensive TSO transactions.
3. Establish separate service classes for Oracle users. Use the service classes to reflect goals and relative importance of different TSO workloads that are classified by user attributes such as userid or accounting information.

CICS TS and IMS TM

CICS TS and IMS TM workloads can be managed using service classes and can be classified using attributes such as userid, transaction name, luname, and subsystem instance name.

Batch

Because batch workloads are typically discretionary in nature, Oracle batch jobs do not need to be separately classified. However, Oracle batch jobs can be distinguished from other batch jobs by establishing separate service classes, as described for the TSO environment.

When Oracle batch jobs are run under a certain service class, consider their priority relative to other Oracle and non-Oracle workloads. In a normal to heavily loaded system, if Oracle batch jobs run at a lower priority than others, the Oracle jobs might be swapped out for lengthy periods. If an Oracle job is swapped out while holding a critical latch, it may adversely impact the performance of other Oracle users.

Special Needs Functions

Special services classes should be considered for privileged users or special jobs such as those described below.

DBA Accounts

The database administrator (DBA) frequently needs priority access to the database in order to perform functions on behalf of all Oracle users. Granting higher relative importance to these types of work shortens the elapsed time for these functions to the benefit of all users. Except for database import and export, DBA functions generally do not require large amounts of Oracle and system resources compared to those of the user community.

Database Imports and Exports

Import and export functions are good candidates for higher relative importance when they involve the entire database. If the performance parameters of your system force swapping among long running batch jobs, you might want to consider non-swappable status for import and export.

Import and export performance can be optimized by maximizing the size of the buffer that is used to transfer rows to and from the export file. The buffer needs to be large enough to hold approximately 1000 table rows to get the best performance from these utilities. In addition, you can improve performance by increasing the number of buffers that are available for reading and writing the export file. Use the DCB BUFNO JCL parameter to increase the number of buffers. The I/O operations that are issued by QSAM and BSAM will not generate a channel program using more than 30 buffers or more than approximately 240 KB. For I/O bound processes such as Export and Import, you should specify a BUFNO that allocates approximately 480 KB of buffers. This value will give you the maximum amount of overlap between two maximum I/O channel programs. Refer to *Oracle Database Utilities* for more information.

Data Loading

The direct path in SQL*Loader is much more efficient than the conventional path. When using the DIRECT option, SQL*Loader is generally I/O bound on the input data file. To reduce the elapsed time that is required for a load operation, you need to increase the number of buffers that are available for reading the input file by adding a DCB BUFNO parameter to the input file allocation. Performance improvements occur as the number of buffers is increased to 200, although 48 buffers yield a significant improvement in the data load rate.

If you cannot use the DIRECT option, then specify the largest bind array size (using the ROWS parameter) that you can. An array size of approximately 1000 rows improves performance significantly over the default size of 64 rows.

Index Creation

The creation of indexes on large tables can consume significant resources. Consider higher relative importance and non-swappable status for these functions.

In addition to z/OS tuning parameters, you need to consider special session settings to support index creation in large tables. Increasing the SORT_AREA_SIZE parameter value can substantially reduce the elapsed times of index creation jobs. This can be done selectively at the session level by using the ALTER SESSION SQL command so that other non-critical jobs will still use the INITORA specified value.

Sorting Data by Key Before Loading

You can use SYNCSORT, DFSORT, or another z/OS sort utility to sort the data by key before loading it into Oracle. Once the data is sorted, load the data into Oracle and create the index with the NOSORT option. For large data loads, this technique can save significant amounts of time when loading data and creating the index.

Remote Clients

Remote clients that access an Oracle server through the Oracle Net service are dispatched on a lightweight unit of work called an enclave SRB within the Net address space. The performance characteristics of such work can be effectively managed when used with WLM in goal mode. Enclave transactions are managed separately from one another as well as from the Oracle Net address space they run in.

The Net startup option ENCLAVE (CALL|SESS) controls how the database request from the client is handled (described under "PARM"). With ENCLAVE(SESS) specified in the PARM value used at Net startup, classification of the work is done once when a new remote connection is made. Oracle Net presents WLM with attributes for workload classification. Some of the network specific attributes that can be used for classification include protocol, host name, or IP address. The list of WLM attributes available for classification is shown in Table 15–2, " Workload Manager Attributes and Values". The enclave will be deleted at session termination (logoff) time. Because the classification happens only once per session, only velocity goals are appropriate for the enclave's service class.

If ENCLAVE(CALL) is specified in the PARM value used at Net startup, then the enclave is deleted when the request from the client is finished (when Net needs more data from the client). Deleting the enclave reports the transaction completion to WLM, providing response time and transaction counts to any workload monitors such as RMF. The next request arriving from the client will be classified into a new enclave. The values available for classification are the same as with ENCLAVE(SESS) above, and are shown in Table 15–2, " Workload Manager Attributes and Values". Because the classification is done for each network request, response time goals should be used for the enclave's service class.

Table 15–2 Workload Manager Attributes and Values

ATTRIBUTE	VALUE
Subsystem Type	C'OSDI'
SI	OSDI subsystem name, for example, WFM1
UI	User ID from connect
NET	first eight characters of dotted IP address (example: 100.024.)
LU	last seven characters of dotted IP address (example: 020.003)
CT	Protocol from connect 'TCP'
SPM	position 1-8: Oracle database service name
SPM	position 9-89: TCP/IP hostname (left justified)

Note: Leading zero characters must be used in the nodes of the dotted IP address.

"Subsystem Type" is not strictly an attribute. WLM has several predefined subsystem types (JES for example). You must define a new subsystem type of "OSDI" to WLM if you desire WLM monitoring of OSDI work. Refer to the IBM document, *MVS Planning: Workload Management*, for information on how to do this, and for information on how to utilize the attributes listed above to manage work. Generally, WLM is configured using ISPF and the IWMARIN0 REXX exec.

As z/OS 1.4 is a pre-requisite for Oracle Database 10*g*, the use of WLM goal mode is required and installations must specify a WLM policy. It is highly recommended that you include a section in this policy for the OSDI subsystem.

If you choose to run without a section in the policy for the OSDI subsystem, then the client work will be assigned the service class SYSOTHER, which has a discretionary goal. Performance is likely to be unsatisfactory.

The following is an example of a WLM classification rules ISPF panel:

```
Subsystem-Type  Xref  Notes  Options  Help
--------------------------------------------------------------------------
              Modify Rules for the Subsystem Type          Row 1 to 4 of 4
Command ===> ____________________________________________   SCROLL ===> PAGE
Subsystem Type .  : OSDI        Fold qualifier names?   Y  (Y or N)
Description    .  .  . OSDI SubSystem Type

Action codes:  A=After    C=Copy         M=Move        I=Insert rule
                  B=Before   D=Delete row    R=Repeat   IS=Insert Sub-rule
                                                                More ===>
              -------Qualifier-------------            -------Class--------
Action    Type       Name     Start                    Service     Report
                                             DEFAULTS: ORACLES     ________
 ____  1  SI          ORAC    ___                      ________    ________
 ____  2  NET       010.100   ___                      ________    ________
 ____  3  LU        001.080   ___                      ORACLEM     ________
 ____  3  LU        001.081   ___                      ORACLEH     ________

**********************************BOTTOM OF DATA *******************************
```

This rule assigns the service class ORACLEM to all work arriving from a client at IP address 10.100.1.80, and assigns ORACLEH to all work from the client at IP address 10.100.1.81. Note that the service class ORACLES is assigned as the default service class to Net workloads that cannot be classified by the above rules. It is very important to specify a default service class. Without a default service class, an error in the classification rules could result in no rules matching. In this case, the request will be assigned service class SYSOTHER, which has a discretionary goal. This will result in undesirable performance characteristics.

If ENCLAVE(CALL) is specified in the PARM value at Net startup, you should specify response goals, or percentile response goals for the service classes used by Oracle enclaves. If ENCLAVE(SESS) is specified in the PARM value at Net startup, you should specify velocity goals for the service classes used by Oracle enclaves.

The following shows a sample screen defining a service class with three periods. The first period has a response time goal of 15 ms. at importance 1. This gives short requests high priority access to the CPU. If the request takes more than 50 CPU service units, the enclave is migrated to a second period at importance 3. If the request is still running after 500 service units, it is then migrated to a third period at importance 5. This design of service class goals is only feasible if the ENCLAVE(CALL) parameter is used. It has the advantage of providing fast, high priority response to short requests, while treating longer requests at low, batch-like priorities.

```
 Service-Class  Xref  Notes  Options  Help
 ----------------------------------------------------------------------
Modify a Service Class             Row 1 to 2 of 2
  Command ===> _________________________________________________________

  Service Class Name . . . . . : ORACLEH
  Description  . . . . . . . . . Oracle Mid Tier #1
  Workload Name  . . . . . . . . ORACLE    (name or ?)
  Base Resource Group  . . . . . ________  (name or ?)

  Specify BASE GOAL information.  Action Codes: I=Insert new period,
  E=Edit period, D=Delete period.

        ---Period---  ---------------------Goal---------------------
 Action  #  Duration   Imp.  Description
   __
   __    1  50         1    Average response time of 00:00:00.015
   __    2  500        3    Average response time of 00:00:00.500
   __    3             5    Execution velocity of 10
 ****************************** Bottom of data *********************
```

16 Error Diagnosis and Reporting

This chapter discusses the diagnosis of suspected Oracle database errors and the requirements for documenting these errors to Oracle Support Services. Specific topics in this chapter include documentation requirements, categorization of errors, system dump requirements, and methods of reporting to Oracle Support Services.

For information on z/OS-specific error messages, refer to the *Oracle Database Messages Guide for IBM z/OS.*

The following topics are included:

- Oracle Support Services
- Providing Error Documentation
- General Documentation Requirements
- Error Diagnosis
- System Dumps
- GTF

Oracle Support Services

Oracle Support Services acts as the interface to the Oracle database user community. Refer to the applicable Oracle Support Services publications for a discussion of policies and procedures for using their services.

Providing Error Documentation

During the error resolution cycle, Oracle Support Services might request you provide them with machine readable data. Send machine readable data, not formatted or printed data. Use FTP (File Transfer Protocol) if you plan to send large amounts of data.

If you are requested to send data to Oracle Support Services, then follow the documentation requirements provided in the section "General Documentation Requirements". Failure to follow these requirements might result in the inability to process your information. This could delay the resolution of any errors you are reporting.

General Documentation Requirements

When you report a suspected error, you might be asked to describe the Oracle database subsystem and z/OS operating system environments in detail. Provide the full version number of each component that has an error. The full version number includes important PUT levels for your z/OS system.

Before you contact Oracle Support Services, ensure this information is available:

- Oracle database library naming conventions
- Method of accessing the Oracle database utilities (batch, TSO, or z/OS UNIX System Services)
- Oracle database subsystem name
- Full version of the Oracle database kernel
- Full version of the Oracle database utility
- Release level of z/OS
- PUT level
- RMID of any relevant OS module

In addition to describing the Oracle database operational environment, detailed documentation specific to the error might be required. This might include:

- Console logs
- Utility SYSOUT
- Utility input files
- System diagnostic messages
- Oracle database error messages
- System dumps

- Database engine trace data sets

Keep in mind that often more than one error is associated with a single failure. Describe all errors for the failure being reported. If your application uses Pro*C, Pro*COBOL, or another Oracle database Precompiler, then ensure your application displays or prints out all errors it encounters. Without a complete and consistent set of information, diagnosing the problem can be impossible.

Error Diagnosis

When investigating a potential Oracle server error, start by determining which component is failing, where it is failing, and the error category.

Components

When reporting a problem to Oracle Support Services, identify the component suspected of failure, along with its full version and correct release level. A list of components and their version numbers is documented in the *Oracle Database Installation Guide for IBM z/OS* and any maintenance tape release bulletin.

Error Categories

Use these error categories to describe the error:

- Documentation
- Incorrect output
- Oracle database external error
- Abend
- Program loop
- Performance
- Missing functionality
- Wait state

Documentation

When reporting documentation errors, you are asked to provide this information:

- Document name
- Document part number

- Date of publication
- Page number

Describe the error in detail.

Documentation errors can include both erroneous documentation and omission of required information.

Incorrect Output

In general, an incorrect output error exists whenever an Oracle database utility produces a result that differs from written Oracle documentation. When describing errors of incorrect output, you need to describe, in detail, the operation of the function in error. Be prepared to describe your understanding of the proper function, the specific Oracle documentation that describes the proper operation of the function, and a detailed description of the incorrect operation.

If you think you have found a software bug, then be prepared to answer these questions:

- Does the problem occur in more than one Oracle tool? (Examples of Oracle tools are SQL*Plus and Oracle Developer/2000).
- What are the exact SQL statements used to reproduce the problem?
- What are the full version numbers of the Oracle database and related Oracle software?
- What is the problem and how is it reproduced?

Oracle Database External Error

Oracle database error messages are produced whenever an Oracle database utility or the Oracle database kernel detects an error condition. Depending on the circumstances, error messages might be fatal or nonfatal to the utility or kernel.

Be prepared to identify the exact error message and message number received and the complete circumstances surrounding the error.

Abend

Any program check in an Oracle database utility or the Oracle database kernel address is considered an error. A system dump is required as documentation in the event of a program check.

Ensure the system dump contains all of the private area of the Oracle database address space. Without it, diagnosis is sometimes impossible.

System abends might or might not indicate a failure of the Oracle database subsystem depending upon circumstances.

The following abends are not considered Oracle database failures:

- 013 - open failure
- 122 - cancelled by operator
- 222 - cancelled by operator
- 322 - CPU time exceeded
- 722 - SYSOUT lines exceeded

Program Loop

A program loop is evident when the Oracle server task consumes CPU time, but no actual work is performed. This situation is substantially different from an Oracle server task that performs most of its operations in cache (also known as a CPU bound job). CPU bound operations might include large batch sorts, sort merge joins, nested loop joins where the driving table is small enough to fit into the SGA, and so forth.

Any program loop that occurs within an Oracle server or utility address space is considered an Oracle database failure. Loop conditions are rarely experienced and are considered serious errors. The initial diagnostic approach with a loop consists of a system dump. If a task is in a program loop, then ensure the system dump includes all of the private area of the Oracle database address space.

Further diagnosis might be required using z/OS SLIP commands. Oracle Support Services furnishes specific instructions on the use of SLIP depending upon circumstances.

Performance

Oracle database system performance is determined by many factors, most of which are not within the control of Oracle Corporation. Considerations such as system load, I/O topology, and database design make the documentation of performance errors difficult.

Provide detailed information about the state of your environment when reporting an error.

Specific documentation might include:

- CPU type and memory configuration

- Database topology
- I/O topology
- System workload by type
- Oracle database workload characterization
- Query execution plans

Missing Functionality

Enhancement requests can be opened with Oracle Support Services to request the inclusion of functions and features that Oracle products do not currently have. When opening an enhancement request, describe the specific feature or function to be added to the product and provide a business case to justify the enhancement.

Wait State

A wait state occurs whenever a required system resource (for example, an enqueue) is unavailable. Because the task requiring the resource is in a blocked or wait state, little or no CPU time is consumed. However, wait states are not limited to a lack of operating system resources. A wait state can occur within the Oracle server due to an incompatible lock request, high contention on an internal latch, an archive task that halts, and so forth.

A wait state in an Oracle database utility might or might not be considered an Oracle database error. A wait state that occurs due to operating system resource conflicts (for example, multiple requests for a tape mount) is not considered an Oracle database error. If a wait state occurs within an Oracle database utility, then a system dump is required of both the utility and kernel address spaces. If a wait state occurs in the Oracle server address space, then a system dump is required of that address space.

Diagnosing Wait State Problems

Before you contact Oracle Support Services, try these recommended approaches:

- Check the Oracle database alert log and system console logs for any error messages.
- Query the V$LOCK dynamic view to determine whether there are any lock conflicts.
- For a high volume online transaction processing (OLTP) system, query the V$WAITSTAT dynamic view to determine whether there is contention for a

class of data blocks. To perform this query, invoke SQL*Plus and enter these commands:

```
SQL> CONNECT / AS SYSDBA
SQL> SELECT * FROM V$WAITSTAT;
```

For more information about the V$WAITSTAT view, refer to the *Oracle Database Reference*.

- If your database is in ARCHIVELOG mode, then ensure your archive data set destination is not full. If the destination is full, then the archiver task cannot copy any more full redo log data sets, thus preventing any further changes to the database. In this case, all tasks attempting to make changes are eventually placed in a wait state.

If you cannot diagnose the wait state problem using the previous suggestions, then first obtain these dumps and contact Oracle Support Services:

- An SVC system dump that includes all of the private area of the Oracle database address space.
- A SYSTEMSTATE dump from within Oracle database. When you contact Oracle Support Services, tell them you performed this task.

  To obtain a SYSTEMSTATE dump, invoke SQL*Plus and perform these commands:

  ```
  SQL> CONNECT  / AS SYSDBA
  SQL> ALTER SESSION SET EVENTS
  'immediate trace name systemstate level 10';
  ```

  A SYSTEMSTATE dump can be quite large, depending on the number of concurrent connections to the Oracle server, the number of resources each connection holds or requests in any mode, and the size of the SGA. The current cursor and object state for each concurrent connection is also included in the SYSTEMSTATE dump. The size of these dumps can easily be in the multiple of megabytes. A SYSTEMSTATE dump is synonymous with a SYSTEMSTATE trace data set.

System Dumps

When providing documentation on suspected Oracle database failures, it might be necessary for you to provide a system dump of the Oracle server or utility address spaces. Dumps are initiated through the z/OS operator interface using the DUMP and SLIP commands, or automatically by Oracle database if it detects a problem.

Dumps sent to Oracle Support Services as documentation for suspected errors must not be formatted. Formatted dumps will not be accepted.

When specifying dump parameters in response to a z/OS DUMP COMM=(' ') command, you must include this specification:

```
PSA,TRT,RGN,CSA
```

Additional parameters may be required.

System Dump Data Sets

Once a SYS1.DUMP*xx* data set is created, the system operator is notified whenever a dump to that data set occurs. Because all Oracle database abends are dumped to SYS1.DUMP data sets and are not dynamically allocated, you must ensure a SYS1.DUMP data set is always available.

You must also ensure the SYS1.DUMP data set is large enough to accommodate a dump of two address spaces (the Oracle database address space and the client address space). This allows for a complete dump if an Oracle database utility abends while in cross memory mode. Refer to "Oracle Server Storage Requirements" in Chapter 15, "Oracle Database Performance", for more information about estimating the size of an Oracle database address space.

If a SYS1.DUMP data set is not available, then a dump might be lost.

Operator Initiated Dumps

Operator initiated dumps are accomplished with the z/OS DUMP command, where *text* is the title you want the dump to have:

```
DUMP COMM=(text)
```

After the DUMP command has been issued, you must respond to the system WTOR with the following command:

```
R xx,[JOBNAME=(jobn)|ASID=(nnn),]SDATA=(PSA,TRT,RGN,CSA)
```

where:

Table 16–1 Variable Descriptions for Code Example

Variable	Description
xx	is the reply identification number.

Table 16–1 (Cont.) Variable Descriptions for Code Example

Variable	Description
jobn	is the name of the started task or batch job.
nnn	is the hexadecimal address space identifier of the address space you want to dump.

New operands can be requested.

SLIP

Proper documentation of an Oracle database error might require the setting of a SLIP trap. If this is the case, then contact Oracle Support Services for specific instructions.

TSO System Dumps

TSO symptom dumps provide a preliminary view of an abend condition in an Oracle database utility. A symptom dump is available by issuing the TSO command:

```
PROFILE WTPMSG
```

The effect of this command is permanent until you issue the command:

```
PROFILE NOWTPMSG
```

GTF

You might need to use GTF as a diagnostic tool under certain circumstances. Oracle Support Services provides specific instructions if this is the case.

17 Migration and Upgrade Considerations

This chapter provides z/OS-specific upgrade information and is intended to be used in conjunction with *Oracle Database Upgrade Guide*.

The following topics are included:

- Requirements
- Upgrade Eligibility
- Preparing to Upgrade
- Upgrading the Database
- OSDI Changes in Oracle9i Release 2

Requirements

In order to upgrade your Oracle Database for z/OS to 10*g* release 2, the z/OS system must be V1.4 or later. This requirement also applies to client applications, including tools, utilities, Access Managers and customer applications.

Upgrade Eligibility

Direct upgrade is supported on z/OS for OSDI-based Oracle8*i* release 8.1.7.4 or Oracle9*i* release 2. Depending on your current release, you may need to upgrade to an interim level prior to upgrading to 10*g* release 2. If upgrading from Oracle8*i* release 8.1.7.4, review the the section "OSDI Changes in Oracle9i Release 2" on page 17-8.

Preparing to Upgrade

The information in this section supplements Chapter 2, "Preparing to Upgrade" in the *Oracle Database Upgrade Guide*.

The Database Upgrade Assistant is not supported on z/OS. A manual upgrade is required for the upgrade method. Refer to the section "Upgrade the Database Manually" in Chapter 3 of the *Oracle Database Upgrade Guide*.

Upgrading the Database

The information in this section supplements Chapter 3, "Upgrading a Database to the New Oracle Database 10*g* Release" in the *Oracle Database Upgrade Guide*.

Install the Database

For instructions on how to install Oracle Database for z/OS 10*g* release 2, refer to the *Oracle Database Installation Guide for IBM z/OS (OS/390).*

Analyze the Database

Use the Pre-Upgrade Information tool as described in the section "Analyze the Database to be Upgraded" in Chapter 3 of the *Oracle Database Upgrade Guide*.

When you install Oracle Database for z/OS, SQL scripts and sub-scripts are installed as members of a PDS. The following batch JCL example uses the z/OS FNA feature provided with most Oracle tools and utilities to allow the SQL scripts and sub-scripts to be read as members of a PDS:

```
//PLUS EXEC PGM=SQLPLUS,REGION=0M,PARM='/nolog'
//STEPLIB DD DISP=SHR,DSN=ORACLE.V10G.CMDLOAD
//ORA$LIB DD DISP=SHR,DSN=ORACLE.V10G.MESG
//ORA$FNA DD *
FSA (FTYPE(SQL) FNAME('//DD:SQL(+)'))
FSA (FTYPE(PLB) FNAME('//DD:SQL(+)'))
//ORA@ORA1 DD DUMMY <-- OSDI SID is 'ORA1'
//SYSIN DD *
CONNECT / AS SYSDBA
@utlu101i
/*
```

Back Up the Database

This section describes the z/OS-specific backup tasks to perform before upgrading you database.

Shut Down the Prior Oracle Instance Cleanly

To make sure your Oracle instance is shut down cleanly, perform a SHUTDOWN NORMAL (or IMMEDIATE) and allow it to complete. If it cannot complete because user sessions remain connected, perform a SHUTDOWN ABORT or forcibly terminate the OSDI service. Then, perform a fresh STARTUP followed by SHUTDOWN NORMAL. It is important for SHUTDOWN NORMAL to complete, ensuring that no outstanding redo data remains in the database. Finally, stop the associated service with an OSDI STOP command or a z/OS STOP (P) command to cause its address space(s) to terminate.

Perform the Backup

This step is recommended, though not mandatory, if you are upgrading from Oracle8*i* release 8.1.7.4 and Oracle9*i* release 2. It provides an extra measure of assurance that no data can be lost in the upgrade process.

Back up the entire database, including control files, online log files, and data files. This can be done with a fast physical data mover such as IBM's DF/DSS. In the following batch JCL example, DF/DSS is used to back up an Oracle7 database, where all of the Oracle7 database files begin with"ORACLE.ORA1":

```
//BACKUP EXEC PGM=ADRDSSU
//SYSPRINT DD SYSOUT=*
//DUMPDS DD DSN=ORACLE.ORA1BK.FULL,
// DISP=(NEW,CATLG,DELETE,)
// UNIT=3480,
// VOL=(,,,99,SER=(vol001,vol002,vol003....)),
// LABEL=(1,SL,EXPDT=98000)
//SYSIN DD *
DUMP DATASET(INCLUDE(ORACLE.ORA1.**) ) -
OUTDD(DUMPDS)
/*
```

Upgrade the Database

This section contains information about how to upgrade the database.

Step 1: Prepare to Start the New Oracle Database Release

In this step, you perform a STARTUP of the new release of Oracle Database. If the associated OSDI database service is still active, stop it with an OSDI STOP command or a z/OS STOP (P) command. After the service terminates, you must modify the service to use the new release code. There are two ways to do this, as follows:

- Modify the existing JCL procedure for the service to specify the new release library in STEPLIB.
- Create a new JCL procedure for the service using a different name from that of the prior release, and use the OSDI ALTER SERVICE command to switch the service to the new procedure. If you choose this option, make sure that the new procedure will be assigned the correct z/OS userid when started. If in doubt, talk to your system security administrator.

It also is possible to define a new OSDI service (specifying a new JCL procedure) and use this to start up the new release. However, the new service cannot have the same SID as the prior release (assuming both are on the same z/OS system) because SIDs must be unique. In most cases you will want the new release to run using the same SID as the prior release.

Make sure that your `init.ora` parameter file for the new release is ready. Refer to the section "Oracle Initialization Parameter Considerations" in Chapter 3, "Configuring a Database Service and Creating a New Database" for new or changed parameters. Also, evaluate the information obtained from the Pre-Upgrade Information tool and refer to the section "Upgrade the Database Manually" in Chapter 3 of the *Oracle Database Upgrade Guide.*

Step 2: Start the New Oracle Database Release

First, start the OSDI database service with an OSDI START command. Make sure the service initializes successfully, signified by message MIR0002I in the system log.

Next, perform an Oracle STARTUP using SQL*Plus from the new release. You can do this from TSO, from a UNIX System Services shell, or by using a batch job. The examples provided here use batch jobs and identify the target instance with an ORA@*`sid`* DD statement. This DD supplies the OSDI SID. However you choose to run SQL*Plus, if you have configured OSDI security features the associated z/OS userid must have the required authorizations: it must be authorized to bind to the database service and it must be able to CONNECT using AS SYSDBA. The following is a batch JCL example:

```
//PLUS EXEC PGM=SQLPLUS,REGION=0M,PARM='/nolog'
//STEPLIB DD DISP=SHR,DSN=ORACLE.V10G.CMDLOAD
```

```
//ORA$LIB DD DISP=SHR,DSN=ORACLE.V10G.MESG
//INITORA DD DISP=SHR,DSN=ORACLE.ORA1.PARMLIB(INITORA1)
//ORA@ORA1 DD DUMMY <-- OSDI SID is 'ORA1'
//SYSIN DD *
CONNECT / AS SYSDBA
STARTUP PFILE='//DD:INITORA' UPGRADE
/*
```

Refer to the section "Upgrade the Database Manually" in Chapter 3 of the *Oracle Database Upgrade Guide,* for common errors that may occur when attempting to start up the new Oracle Database release.

Step 3: Create a SYSAUX Tablespace

Refer to the section "Upgrade the Database Manually" in Chapter 3 of the *Oracle Database Upgrade Guide,* for the details on the CREATE TABLESPACE command.

The following batch JCL example creates a SYSAUX tablespace:

```
//PLUS EXEC PGM=SQLPLUS,REGION=0M,PARM='/nolog'
//STEPLIB DD DISP=SHR,DSN=ORACLE.V10G.CMDLOAD
//ORA$LIB DD DISP=SHR,DSN=ORACLE.V10G.MESG
//INITORA DD DISP=SHR,DSN=ORACLE.ORA1.PARMLIB(INITORA1)
//ORA@ORA1 DD DUMMY <-- OSDI SID is 'ORA1'
//SYSIN DD *
CONNECT / AS SYSDBA
CREATE TABLESPACE ...
/*
```

Step 4: Run Upgrade Scripts

In this step, you run SQL scripts to upgrade the Oracle dictionary structures to the new release. The scripts are provided in the .SQL data set installed with the new release. The choice of scripts to run depends on your prior Oracle Database release (the one from which you are upgrading). Refer to the section "Upgrade the Database Manually" in Chapter 3 of the *Oracle Database Upgrade Guide* to determine which script(s) you must run.

The following example JCL for a batch SQL*Plus job executes the upgrade script to complete a migration from Oracle9*i*, R2:

```
//PLUS EXEC PGM=SQLPLUS,REGION=0M,PARM='/nolog'
//STEPLIB DD DISP=SHR,DSN=ORACLE.V10g.CMDLOAD
//ORA$LIB DD DISP=SHR,DSN=ORACLE.V10g.MESG
//SQL DD DISP=SHR,DSN=ORACLE.V10g.SQL
//ORA@ORA1 DD DUMMY <-- OSDI SID is 'ORA1'
```

```
//ORA$FNA DD *
FSA (FTYPE(SQL) FNAME('//DD:SQL(+)'))
FSA (FTYPE(PLB) FNAME('//DD:SQL(+)'))
/*
//SYSIN DD *
CONNECT / AS SYSDBA
@U0902000
/*
```

Examine the SQL*Plus output from these scripts and the Oracle instance alert log for significant error indications. Certain DROP statements in the scripts may produce "object not found" errors which are not considered significant. Other errors, particularly ORA-0060x (internal) errors, must be resolved before proceeding.

Step 5: Run the utlu101s.sql Script

The Post-Upgrade Status tool displays the status of the database components in the upgraded database. Refer to the section "Upgrade the Database Manually" in Chapter 3 of the *Oracle Database Upgrade Guide* for more information.

The following batch JCL example runs the `utlu101s.sql` script:

```
//PLUS EXEC PGM=SQLPLUS,REGION=0M,PARM='/nolog'
//STEPLIB DD DISP=SHR,DSN=ORACLE.V10g.CMDLOAD
//ORA$LIB DD DISP=SHR,DSN=ORACLE.V10g.MESG
//SQL DD DISP=SHR,DSN=ORACLE.V10g.SQL
//ORA@ORA1 DD DUMMY <-- OSDI SID is 'ORA1'
//ORA$FNA DD *
FSA (FTYPE(SQL) FNAME('//DD:SQL(+)'))
FSA (FTYPE(PLB) FNAME('//DD:SQL(+)'))
/*
//SYSIN DD *
CONNECT / AS SYSDBA
@utlu101s.sql TEXT
/*
```

Step 6: Shut Down and Restart the Instance

This step is required to perform internal housekeeping tasks. At the end of the STARTUP process, your database is migrated or upgraded and is usable. However, you should perform a STARTUP without RESTRICT, to enable normal client access, as shown in the following batch JCL example:

```
//PLUS EXEC PGM=SQLPLUS,REGION=0M,PARM='/nolog'
//STEPLIB DD DISP=SHR,DSN=ORACLE.V10G.CMDLOAD
//ORA$LIB DD DISP=SHR,DSN=ORACLE.V10G.MESG
```

```
//INITORA DD DISP=SHR,DSN=ORACLE.ORA1.PARMLIB(INITORA1)
//ORA@ORA1 DD DUMMY <-- OSDI SID is 'ORA1'
//SYSIN DD *
CONNECT / AS SYSDBA
SHUTDOWN IMMEDIATE
STARTUP PFILE='//DD:INITORA'
/*
```

Step 7: Recompile Database Stored Procedures

Some stored procedures in the database will have been invalidated during migration or upgrade due to changes in underlying database objects. Although these will recompile dynamically as needed, Oracle recommends forcing the recompiles. This involves running a supplied script to verifiy that all expected packages and classes are valid.

Refer to Refer to the section "Upgrade the Database Manually" in Chapter 3 of the *Oracle Database Upgrade Guide* for more information.

The following batch JCL example recompiles the database stored procedures:

```
//PLUS EXEC PGM=SQLPLUS,REGION=0M,PARM='/nolog'
//STEPLIB DD DISP=SHR,DSN=ORACLE.V10G.CMDLOAD
//ORA$LIB DD DISP=SHR,DSN=ORACLE.V10G.MESG
//SQL DD DISP=SHR,DSN=ORACLE.V10G.SQL
//ORA@ORA1 DD DUMMY <-- OSDI SID is 'ORA1'
//ORA$FNA DD *
FSA (FTYPE(SQL) FNAME('//DD:SQL(+)'))
FSA (FTYPE(PLB) FNAME('//DD:SQL(+)'))
/*
//SYSIN DD *
CONNECT / AS SYSDBA
@UTLRP
SELECT count(*) from dba_objects WHERE status='INVALID';
SELECT distinct object_name FROM dba_objects WHERE status='INVALID';
/*
```

If you encounter problems, refer to the section "Troubleshooting the Upgrade" in Chapter 3 of the *Oracle Database Upgrade Guide*.

Fallback and Recovery Considerations

If you encounter problems with the new Oracle Database release, you may need to fall back (return) to your prior Oracle Database release. To fall back, perform a

SHUTDOWN of the newly installed release instance, if possible, and STOP the OSDI service, waiting for all service address spaces to terminate.

Refer to Chapter 7, "Downgrading a Database Back to the Previous Oracle Database Release" in the *Oracle Database Upgrade Guide* for information on using the backup database created prior to the upgrade, or, if you have used the COMPATIBLE database parameter, the sequence of scripts to be used to downgrade the database.

After Upgrading the Database

Refer to Chapter 4, "After Upgrading the Database" in the *Oracle Database Upgrade Guide* for information on tasks and the asociated scripts to run after upgrading a database. The scripts can be run from TSO, from a UNIX Systems Services shell, or by using a batch job, modifying the sample SQL*Plus jobs provided in this section.

OSDI Changes in Oracle9*i* Release 2

Oracle9*i*, R2 introduces several z/OS architectural changes described here.

Network Client Operations

Starting with Oracle9*i*, R2, z/OS Oracle client programs that use Oracle Net open network protocol connections directly rather than using the Oracle Net service. This affects customer-written Oracle applications running in CICS TS, TSO, and batch and Oracle tools or utilities running in TSO and batch. It also affects situations where a z/OS database instance operates as a "network client", including opening network database links and connections for the UTL_HTTP PL/SQL package. It does not affect Oracle applications, tools, or utilities running in a UNIX System Services shell environment: these programs are already using network protocol services directly.

Previously, the network protocol interaction for applications not running in a UNIX System Services shell environment was executed by the network service address space on the client's behalf. Now it is executed directly from the client's z/OS address space and TCB. This change is expected to improve performance and eliminate client operational dependence on the network service. It has the following external considerations:

- **OMVS dubbing**. IBM's TCP/IP protocol makes use of UNIX System Services. A z/OS address space that uses IBM TCP/IP must therefore be a UNIX System Services process or be capable of being "dubbed". Dubbing occurs automatically when needed but it requires the z/OS userid associated with the address space to have a default UNIX System Services segment defined in the security subsystem (e.g. RACF). If dubbing fails, the network connection will not be opened and the application probably will fail. You must ensure that all z/OS clients that connect to remote Oracle servers can be dubbed. If in doubt, discuss this with your z/OS security administrator.

- **No client dependence on network service**. With this change, stopping and starting the Oracle network service has no effect on z/OS client network connections. z/OS clients no longer require a TNS@*xxx* DD statement or other specification o identify a network service to use, and they no longer require OSDI bind authority to the network service. The network service is now involved only with inbound network operations (clients on other systems who are connecting to a z/OS database instance).

Clients Use Language Environment

Beginning with Oracle9*i*, R2, all Oracle client software on z/OS, including Oracle tools, utilities, and the Oracle program interface code used by customer applications, uses IBM Language Environment (LE) for C program runtime services. LE replaces the MVS Oracle C runtime library in use since 1986. This change is primarily internal in nature but it does affect a few externals, including the syntax used to specify files and certain PARM and command line features. It also requires that LE runtime (e.g. SYS1.SCEERUN) be available in the system linklist or JOBLIB/STEPLIB when running Oracle tools, utilities, or customer-written applications.

Detailed information on this change is provided in the *Oracle Database User's Guide for IBM z/OS*. Since Oracle database administration involves running Oracle tools and utilities, refer to the *Oracle Database User's Guide for IBM z/OS* as necessary when performing administration tasks discussed in this manual.

Configuring Network Service

Beginning with Oracle9*i*, R2, the OSDI network service is involved only in inbound network operations, that is, when remote clients connect to a z/OS Oracle instance. Outbound clients on z/OS--ones who are connecting to remote Oracle instances--interact directly with the TCP/IP protocol rather than using the network service. This is true for all Oracle9*i*, R2 clients: TSO, batch, UNIX System Services, Access Managers, and even the Oracle database server when it operates as a client (for example, when opening database links). You do need to run OSDI network service for inbound client support and in some cases for providing backward compatibility.

The OSDI network service is simple to configure. You must issue an OSDI DEFINE SERVICE command with TYPE(NET) to create the OSDI data structures used to manage the service. As with the database service, a JCL procedure must be installed in a system procedure library at some point before you attempt to start the service. Rather than reading a parameter data set, network service region parameters are specified directly in the PARM string of DEFINE SERVICE. Only TCP/IP is supported. A port number must be supplied; this is the port on which the service listens for inbound remote clients.

Beginning with Oracle9*i*, R2, the SID of the network service is irrelevant because outbound clients do not use the network service. If you have Oracle8*i* or Oracle9*i*, R1 clients making outbound network connections, the network service SID is significant since those clients must identify the network service they want to use.

As discussed earlier, OSDI database servers require no specific action or configuration to be accessible to network clients. All remote clients specify the same hostname (or IP address) and port number in the Oracle network address string or tnsnames.ora entry. The target database instance is specified using the SID parameter of CONNECT_DATA in the address string.

Configuring Oracle Access Managers

Oracle Access Managers for Oracle9*i*, R2 can access an Oracle9*i*, R2 local or remote database server. When accessing a remote server, Oracle Access Managers utilize the Oracle Net open network protocol with the OMVS dubbing requirement described under "Network Client Operations" on page 17-8.

In the case where a local database server is upgraded to Oracle9*i*R2, earlier versions of Oracle Access Managers can access the local database if the connection in the thread table (CICS) or RTT table (IMS TM) is changed from a local definition to a remote definition using TCP/IP.

Oracle Access Manager for CICS TS

Oracle9*i*, R2 Access Manager for CICS TS thread definitions remain consistent with earlier versions. To access this version, customer applications must be re-linked with a new application stub (ORACSTUB). Refer to Chapter 11, "Oracle Access Manager for CICS TS" for more information.

A

OSDI Subsystem Command Reference

OSDI provides a set of system commands for defining and controlling instances of Oracle products. This appendix is the primary reference for all OSDI commands.

The topics in this appendix include:

OSDI Command Reference

OSDI provides a set of system commands for defining and controlling instances of Oracle products. This appendix is the primary reference for all OSDI commands.

Command Types and Processing

OSDI commands are broadly divided into definition commands and operating commands, as follows:

- Definition commands are used to create and manipulate data structures that describe service groups and services. These commands commonly appear in the subsystem configuration file.
- Operating commands are used to manage execution of services.

Three mechanisms exist for issuing OSDI commands:

- Subsystem configuration file, processed during subsystem initialization
- System operator command interfaces for z/OS (system consoles, SDSF, SYS1.PARMLIB (COMMND*xx*), Netview, and others)
- OSDI program interface, used internally by Oracle products.

When an OSDI command is issued using a z/OS system operator command interface, the target subsystem is specified by using the command prefix associated with the subsystem. When the program interface is used to issue an OSDI command, the target is identified by its subsystem name rather than a command prefix. Commands in the configuration file always apply to the subsystem (service group) being initialized, and they must omit the prefix.

Commands generally result in synchronous response messages ranging from simple acknowledgment to multiline displays. Responses to commands that are issued via system operator facilities normally are directed to the issuing console. Various operating commands can result in subsequent, asynchronous messages. These messages are not necessarily directed to the console or session that issued the original command.

System Symbols in Commands

In order to meet the requirement that the particulars of a service (that runs on multiple systems in a sysplex) can be tailored by system, z/OS system symbols (alphanumeric names prefixed with the ampersand character, "**&**") can be used in the specification of certain OSDI command parameters. These command parameters resolve to system-specific or IPL-specific values set by z/OS or by the installation. If a command parameter can include system symbols, this capability is noted in the parameter description.

Definition Commands

Definition commands are used to create, modify, and display the data structures of the service group. An initial set of commands in the configuration file directs the building of these structures during subsystem initialization. Subsequent definition commands can be used to add new service definitions, modify existing definitions, and so forth. The overall data structure persists for the life of the IPL.

The definition commands operate only on data structures; they do not directly affect the operation of services.

Three definition commands are supported:

- DEFINE - Create a logical structure.
- ALTER - Modify the definition of an existing logical structure.
- SHOW - Display contents of an existing logical structure.

Note: **SHOW** deals with definition data only and is distinct from the operating command, **DISPLAY**. Refer to "SHOW" on page A-5, "SHOW" on page A-10, and "DISPLAY" on page A-12.

Structures

The definition commands operate on the following structures:

- SERVICEGROUP - The primary structure of the subsystem.
- SERVICE - A structure representing an instance of an Oracle product.

The various parts of these structures are generally referred to as attributes. Definition commands use keywords to identify attributes being set or modified.

Service Group Definition Commands

DEFINE

DEFINE SERVICEGROUP must be the first command issued to a newly-initialized subsystem; normally it appears in the configuration file just after the bootstrap (INIT) record. DEFINE SERVICEGROUP must be processed successfully in order for any other OSDI commands or functions to be usable.

The command structure for defining a service group is shown in the following example:

```
DEFINE SERVICEGROUP
   [ SSID( string ) ]
   [ DESCRIPTION( string ) ]
   [ MODE( LOCAL | SYSPLEX | *SYS ) ]
   [ SYSTEMS( sysname [ sysname... ] | *ALL ) ]
```

Define Parameters

Parameter	Definition
SSID	Specifies the one-character to four-character z/OS subsystem name associated with the service group. If coded, then it must match the subsystem identifier specified in the IEFSSN*xx* parmlib member or the SETSSI operator command. This parameter defaults to the correct value. It is therefore coded only to confirm that the correct configuration file is in use. `String` cannot contain system symbols.
DESCRIPTION	Specifies an arbitrary text string of up to 64 characters that appears in certain displays associated with the service group. This can be used to supply installation-specific identification for the service group. The default value is '`Service Group ssn`' where *ssn* is the subsystem name. `String` can contain system symbols, but should not contain non-printable characters or control characters.
MODE	This parameter is not yet supported.
SYSTEMS	This parameter is not yet supported.

ALTER

ALTER SERVICEGROUP is used to modify attributes of a service group. It can be included in the configuration file and it can be issued after initialization. Not all attributes can be altered. The subsystem ID, for example, is constant for the life of the IPL.

The command structure for altering a service group is shown in the following example:

```
ALTER SERVICEGROUP
    [ DESCRIPTION( string ) ]
    [ MODE( LOCAL | SYSPLEX ) ]
    [ SYSTEMS( sysname [ sysname... ] | *ALL ) ]
```

Alter Parameters

Parameter	Description
DESCRIPTION	Specifies an arbitrary text string of up to 64 characters that appears in certain displays associated with the service group. This can be used to supply installation-specific identification for the service group. *`String`* can contain system symbols but should not contain non-printable or control characters.
MODE	This parameter is not yet supported.
SYSTEMS	This parameter is not yet supported.

SHOW

SHOW SERVICEGROUP is used to display the current definition of the service group. It can be included in the configuration file and it can be issued after initialization. The command for displaying a service group definition is shown in the following example:

```
SHOW SERVICEGROUP
   [ LONG ]
```

Show Parameters

The `LONG` show parameter specifies that the name, type, and description of each service in the service group be included in the display.

Service Definition Commands

DEFINE

DEFINE SERVICE is used to create a structure for executing an installed Oracle product. It can be included in the configuration file and it can be issued after initialization. Once a service is defined, it can be started, stopped, etc. using operating commands.

The command structure for defining a service is shown in the following example:

```
DEFINE SERVICE
     name
     TYPE( string )
     PROC( string )
   [ DESCRIPTION( string ) ]
   [ PARM( string ) ]
   [ MAXAS( number ) ]
   [ SID( string ) ]
   [ JOBNAME( string ) ]
   [ JOBACCT( string ) ]
   [ MODE( SYSPLEX | LOCAL ) ]
   [ SYSTEMS( sysname [ sysname... ] | *SVG ) ]
```

Define Parameters

Parameter	Description
name	Specifies the name of the service. The name is used in other commands that operate on the service or its definition and by program functions that interact with the service. It must be one to eight characters long. (Although OSDI permits service names up to eight characters long, the name you use for a database service should be seven characters or less due to a length limitation on what is stored in the database control file.) In addition, it must consist of upper case alphabetic, numeric, and/or national characters, and must begin with an alphabetic character. It must be unique within the service group. `Name` cannot contain system symbols. **Note:** Unless you specify JOBNAME, `name` is used as the job identifier when the service is started. In this case, `name` must not be the same as any subsystem name in your system.
TYPE	Specifies the Oracle product being configured. `String` is a one-character to four-character alphabetic and/or numeric identifier that designates an installed Oracle for z/OS product managed under this architecture. Current supported values are ORA (Oracle database) and NET (Oracle Net). `String` cannot contain system symbols.
PROC	Specifies the member name in a system JCL procedure library used to start an address space for the service. Usually this is a procedure that is created during installation of the associated Oracle for z/OS product. `String` cannot contain system symbols.
DESCRIPTION	Specifies an arbitrary text string of up to 64 characters that appears in certain displays associated with the service. This can be used to supply installation-specific identification for the service. The default value is '`Service` *`svc`* `Type` *`type`*', where *`svc`* is the service name and *`type`* is the type. `String` can contain system symbols but should not contain non-printable or control characters.

Parameter	Description
PARM	Specifies a parameter string passed to the service when an address space is started. *`String`* can be from 0 characters to 64 characters. If it contains characters other than alphabetic, numeric, and national characters, then it must be enclosed in single apostrophes. To indicate a zero-length (empty) parameter, specify `PARM('')`. Content requirements for this string depend on the service type and are discussed in Chapter 3, "Configuring a Database Service and Creating a New Database" and . The default is a null string. The value specified can contain system symbols.
MAXAS	Specifies the maximum number of address spaces that can be started for the service. This is meaningful only for a database (TYPE(ORA)) service. If specified for any other type, then *`number`* must be 1. For a database service, *`number`* must be between 1 and 255 inclusive. The default for this parameter is 1. The value cannot include system symbols.
SID	Specifies a unique identifier for the service that is used in client- or application-supplied service addressing. *`String`* is a one-character to eight-character identifier that conforms to the same rules as those for service names. The value supplied must be unique within the z/OS image: it cannot duplicate the SID of any other service in this or any other service group. This parameter defaults to the service name. If you accept the default, it means that the service name must be unique within the z/OS image. **Note:** If you migrate an MPM-based database to OSDI and are supporting pre-OSDI local client applications, you probably want SID to match the subsystem name you were using with MPM.
JOBNAME	Specifies a z/OS jobname to use when starting address spaces for the service. *`String`* is either a one-character to eight-character identifier that conforms to z/OS jobname requirements, or it is a one-character to five-character identifier followed by an asterisk. The asterisk form can be used with multiple address space services to provide unique jobnames; it is replaced with a three-digit address space counter (001, 002, and so on) as each address space is started. The jobname should conform to any rules or requirements specific to your installation. This parameter has no default. If you omit it, service address spaces are started without a jobname but with the service name as the address space identifier. System symbols cannot be used in this parameter.

Parameter	Description
JOBACCT	Specifies an installation-specific string containing job accounting fields. *String* can be from 1 character to 64 characters. If it contains characters other than alphabetic, numeric, and national characters, then it must be enclosed in single apostrophes. Use this parameter if your installation requires job accounting data to be included with started tasks (STCs). If this parameter is omitted or specified as a null string (a pair of adjacent apostrophes), then no job accounting data is supplied when service address spaces are started. *String* can include system symbols.
MODE	This parameter is not yet supported.
SYSTEMS	This parameter is not yet supported.

Refer to "Examples" in Chapter 2, "Configuring and Initializing the Subsystem".

ALTER

ALTER SERVICE is used to modify attributes of a defined service. It can be included in the configuration file and it can be issued after initialization. The name, type, maximum address spaces, and SID of a service cannot be altered.

OSDI does not prohibit altering the definition of a running service. This enables some useful capabilities but it may also be harmful if misused. For example, changing the JCL procedure of a running multiple address space service would have unpredictable consequences when additional auxiliary address spaces are started.

The command structure for altering a service is shown in the following example:

```
ALTER SERVICE
      name
    [ DESCRIPTION( string ) ]
    [ PROC( string ) ]
    [ PARM( string ) ]
    [ MAXAS( number ) ]
    [ JOBNAME( string ) ]
    [ JOBACCT( string ) ]
    [ MODE( SYSPLEX | LOCAL ) ]
    [ SYSTEMS( sysname [ sysname... ] | *SVG ) ]
```

Alter Parameters

Parameter	Description
name	Specifies the name of the service to be altered. It must be the name of an existing service in the service group. `Name` cannot contain system symbols. **Note:** The name of a service cannot be altered.
DESCRIPTION	Specifies an arbitrary text string of up to 64 characters that appears in certain displays associated with the service. This can be used to supply installation-specific identification for the service. `String` can contain system symbols but should not contain non-printable or control characters.
PROC	Specifies the member name in a system procedure library used to start an address space for the service. Usually this is a procedure that is created during installation of the associated Oracle for z/OS product. `String` cannot contain system symbols. **Note:** Altering PROC while a multiple address space or multiple system service is running can have unpredictable effects.
PARM	Specifies a parameter string passed to the service when an address space is started. Requirements for this string depend on the service type and are discussed in the associated Oracle for z/OS product documentation. The value specified can contain system symbols. **Note:** Altering PARM while a multiple address space or multiple system service is running can have unpredictable effects.
MAXAS	Specifies the maximum number of address spaces that can be started for the service. This is meaningful only for a database (TYPE(ORA)) service. MAXAS is accepted on an ALTER SERVICE command only when the service is not active. If the service is active, starting, or stopping, an ALTER SERVICE command with MAXAS specified will be rejected.
JOBNAME	Specifies a jobname to use when starting service address spaces, as discussed under DEFINE SERVICE. You can nullify (remove) a service jobname specification by coding JOBNAME(''). The value specified cannot contain system symbols.

Parameter	Description
JOBACCT	Specifies job accounting data to be included when service address spaces are started, as discussed under DEFINE SERVICE. You can nullify (remove) job accounting data by coding JOBACCT(''). The value specified can contain system symbols.
MODE	This parameter is not yet supported.
SYSTEMS	This parameter is not yet supported.

SHOW

The SHOW SERVICE command is used to display the current definition of a service.

The command for displaying a service definition is shown in the following example:

```
SHOW SERVICE
     name
```

Show Parameters

The `name` show parameter specifies the name of the service whose definition is to be displayed. It must be the name of an existing service in the service group. Name cannot contain system symbols.

Operating Commands

Operating commands manage the execution of services. They are normally issued via the z/OS command interface, either automatically (for example, COMMND*xx* member of SYS1.PARMLIB) or by a real operator. They might also be issued through the OSDI program interface by a management component such as Oracle Enterprise Manager Agent. Operating commands are also permitted in the service group configuration file.

Available Commands

Five operating commands are provided:

- START - Start execution of a service.
- DISPLAY - Display operating status of a service.

- DRAIN - Place a service in quiescent state.
- RESUME - Restore a service to normal operation.
- STOP - Stop execution of a service (see note below).

All of the operating commands take a service name as the first positional parameter. This service name must be the name of a defined service.

Commands

START

The START command initiates execution of an address space for a specified service. For a database service, this can be the first address space (service not previously started) or an auxiliary address space (service previously started and initialized successfully but not yet at its maximum address spaces). For other types of services the service must not already be running.

The command structure for starting a service is shown in the following example:

```
START
    name
   [ PARM( string ) ]
```

START Parameters

Parameter	Description
name	*Name* specifies the name of the service to start. It must be a defined service whose current state is **inactive** or, if **active**, must not already have its maximum address spaces running.
PARM	Specifies a parameter string passed to the service when an address space is started. This overrides any PARM value established by DEFINE or ALTER SERVICE. Requirements for this string depend on the service type and are discussed in Chapter 3, "Configuring a Database Service and Creating a New Database" and Chapter 8, "Oracle Net". *String* can contain system symbols. **Note:** A PARM override must not be used when starting auxiliary address spaces for a database service.

DISPLAY

The DISPLAY command displays execution status information for services. OSDI displays the current operating state of the service. If the service state is "active" or "drained", then the command is also posted to the running service for further processing at the service's discretion.

The command structure for displaying a service is shown in the following example:

```
DISPLAY
     name
   [ LONG ]
```

DISPLAY Parameters

Parameter	Description
name	Specifies the name of the service to be displayed.
LONG	Specifies that a more detailed display is desired. This provides information about each active address space of the service.

DRAIN

The DRAIN command places a running service in a quiescent state in which it no longer accepts new connection (bind) requests. Existing connections or sessions are not affected. The command is also posted to the running service for further discretionary processing. This command has no effect when the service is not running.

The command structure for draining a service is shown in the following example:

```
DRAIN
     name
```

DRAIN Parameters

Parameter	Description
name	Specifies the name of the service to be made quiescent. This must be the name of a running service whose current state is **active**.

RESUME

The RESUME command reverses the effect of a drain, allowing a service to begin accepting new connection requests. The command is also posted to the running service for further discretionary processing. This command has no effect when the service is not running.

The command structure for resuming a service is shown in the following example:

```
RESUME
     name
```

RESUME Parameters

Parameter	Description
name	Specifies the name of the service to be resumed. This must be the name of a running service whose current state is **drained**.

STOP

The STOP command requests termination of a running service. The normal mode of stop changes the service state to **stopping** and posts the stop request to the service; it is up to the service to comply, presumably after allowing current requests to complete and performing required cleanup. A **force** option causes the service address space(s) to be terminated involuntarily. The **force** form of stop requires that a normal stop be issued first. This command has no effect when the service is not running.

The command structure for stopping a service is shown in the following example:

```
STOP
     name
   [ FORCE ]
```

STOP Parameters

Parameter	Description
name	Specifies the name of the service to be stopped. This must be the name of a running service whose current state is **active**, **drained**, or (if the FORCE option is specified) **stopping**.
FORCE	Specifies that an involuntary stop is requested.

OSDI Command Keyword Abbreviations

The abbreviated or alternate forms that can be used for OSDI command verbs and parameter keywords are as follows:

ALTER	ALT
DEFINE	DEF
DESCRIPTION	DESC
DISPLAY	DIS, D
DRAIN	DR
FORCE	(none)
JOBACCT	ACCT
JOBNAME	JOB
LONG	L
MAXAS	MXA
MODE	MD
PARM	P
PROCEDURE	PROC
RESUME	RES
SERVICE	SRV, SVC
SERVICEGROUP	SVG, SG
SHOW	SH
SID	IDENTIFIER, ID
SSID	(none)
START	ST, S
STOP	P
SYSTEMS	SYS
TYPE	TY

B
Operating System Dependent Variables

This appendix contains the z/OS-specific allowed, default, and maximum values for Oracle initialization and storage parameters. For a complete discussion of these parameters, refer to *Oracle Database Reference*. To find out which parameters described in *Oracle Database Reference* are supported or not supported by Oracle Database for z/OS, refer to *Oracle Database Installation Guide for IBM z/OS*

The topics in this appendix include:

Initialization Parameters with z/OS-Specific Defaults or Limits

The following table lists the Oracle Database initialization parameters that have z/OS-specific defaults or limits. You can separate the keywords with commas or blanks. For more information, refer to the section "Oracle Initialization Parameter Considerations" in Chapter 3, "Configuring a Database Service and Creating a New Database".

Table B–1 Initialization Parameters with z/OS-Specific Defaults or Limits

Parameter	Default	Maximum	Minimum
ASM_DISKSTRING	None	Not supported	Not supported
AUDIT_FILE_DEST[1]	0 (zero) means "do not write SMF records"	255	128
AUDIT_TRAIL[1]	N/A	N/A	N/A
BACKGROUND_CORE_DUMP	None	Not supported	Not supported
BACKGROUND_DUMP_DEST	None	Not supported	Not supported
BITMAP_MERGE_AREA_SIZE	N/A	Limited by address space size	0
COMMIT_POINT_STRENGTH	1	N/A[2]	N/A
CONTROL_FILES[1]	&ORAPREFD..&ORASRVN.. DFLCNTL.DBF	N/A	N/A
CORE_DUMP_DEST	None	Not supported	Not supported
CPU_COUNT	N/A	N/A	N/A
CREATE_BITMAP_AREA_SIZE	N/A	Limited by address space size	65,536
DB_*n*K_CACHE_SIZE	0 - only 4, 8, 16, and 32 are supported for *n*	Limited by address space size	0
DB_BLOCK_BUFFERS	N/A	Limited by address space size	N/A
DB_BLOCK_SIZE1	4096	32768	4096
DB_CREATE_FILE_DEST[1]	N/A	23	N/A
DB_CREATE_ONLINE_LOG_DEST_*n* [1]	N/A	29	N/A
DB_FILE_MULTIBLOCK_READ_COUNT	Derived	256	1
DB_FILE_NAME_CONVERT[1]	N/A	N/A	N/A

Table B–1 (Cont.) Initialization Parameters with z/OS-Specific Defaults or Limits

Parameter	**Default**	**Maximum**	**Minimum**
DB_FILES	N/A	65,534	N/A
DB_KEEP_CACHE_SIZE	N/A	Limited by address space size	0
DB_RECYCLE_CACHE_SIZE	N/A	Limited by address space size	0
DBWR_IO_SLAVES	0	N/A	N/A
DG_BROKER_CONFIG_FILE*n*	&ORAPREFD.&ORASRVN.DR*n*.DAT	None	None
DISPATCHERS (SESSIONS, CONNECTIONS)	N/A	1	0
HASH_AREA_SIZE	N/A	Limited by address space size	0
INSTANCE_NUMBER	N/A	No limit	0
JAVA_POOL_SIZE	N/A	Limited by address space size	0
LARGE_POOL_SIZE	N/A	Limited by address space size	N/A
LOCK_SGA[1]	None	Not supported	Not supported
LOG_ARCHIVE_DEST[1]	NULL	None	None
LOG_ARCHIVE_FORMAT[1]	"T%T.S%S.R%R"	None	None
LOG_BUFFER	16384	Limited by address space size	N/A
LOG_FILE_NAME_CONVERT[1]	N/A	N/A	N/A
MAX_DISPATCHERS	None	1	0
MAX_DUMP_FILE_SIZE	None	Not supported	Not supported

Table B–1 (Cont.) Initialization Parameters with z/OS-Specific Defaults or Limits

Parameter	**Default**	**Maximum**	**Minimum**
MAX_SHARED_SERVERS	None	Limited by address space size	0
NLS_LANGUAGE	AMERICAN	N/A	N/A
NLS_TERRITORY	AMERICA	N/A	N/A
OBJECT_CACHE_MAX_SIZE_PERCENT	N/A	Limited by address space size	0
OBJECT_CACHE_OPTIMAL_SIZE	N/A	Limited by address space size	10240
OPEN_CURSORS	N/A	Limited by address space size	1
OS_AUTHENT_PREFIX	OPS$	None	None
PARALLEL_EXECUTION_MESSAGE_SIZE	N/A	N/A	N/A
PARALLEL_THREADS_PER_CPU	2	N/A	N/A
PROCESSES	Derived	Limited by address space size	N/A
RECOVERY_PARALLELISM	Number of processors minus 1.	Limited by PARALLEL_MAX_SERVERS	0
SESSION_CACHED_CURSORS	N/A	Limited by address space size	0
SESSION_MAX_OPEN_FILES	N/A	N/A	N/A
SGA_MAX_SIZE1	The actual SGA size (This parameter should not be specified on z/OS)	N/A	N/A
SGA_TARGET	0 (SGA tuning disabled)	N/A	64
SHADOW_CORE_DUMP	None	Not supported	Not supported

Table B–1 (Cont.) Initialization Parameters with z/OS-Specific Defaults or Limits

Parameter	Default	Maximum	Minimum
SHARED_POOL_SIZE	32000000	Limited by address space size	4194304
SORT_AREA_SIZE	N/A	Limited by address space size	N/A
SPFILE[1]	N/A	N/A	N/A
STANDBY_ARCHIVE_DEST[1]	&ORAPREFD..&ORASRVN.. ARCHLOG.	None	None
STREAMS_POOL_SIZE	N/A	Limited by address space size	0
TIMED_OS_STATISTICS	0 (Default is recommended value on z/OS)	N/A	N/A
TRACEFILE_IDENTIFIER[1]	N/A	N/A	N/A
TRANSACTIONS_PER_ROLLBACK_SEGMENT	5	34	N/A
USER_DUMP_DEST	None	Not supported	Not supported

[1] More information on using this parameter can be found in the section "Oracle Initialization Parameter Considerations" in Chapter 3, "Configuring a Database Service and Creating a New Database".

[2] N/A means that the platform-specific value is the same as the generic value.

Database Limits

Oracle database files, with exceptions, are limited to 4 GB each in size. The exceptions are as follows:

- Control files, which are limited to 20,000 logical blocks of size DB_BLOCK_SIZE
- Bigfile tablespace files, which are limited to the lesser of 4 GB blocks of size DB_BLOCK_SIZE, or, the maximum size of an extended-format extended addressability data set (up to 59 volumes or 16, if striping). For more information, refer to the IBM document *DFSMS: Using Data Sets*.

A maximum of 255 redo log files can be specified for a database.

A database can contain up to 65, 534 datafiles over all tablespaces. A single smallfile tablespace is limited to 4094 datafiles.

SQL Language Parameters

This section lists the parameters for the CREATE CONTROLFILE and CREATE DATABASE SQL statements that have z/OS-specific defaults or limits.

CREATE CONTROLFILE

The parameters for the CREATE CONTROLFILE SQL statement are listed in the following table:

Table B–2 CREATE CONTROLFILE Parameters with z/OS-Specific Defaults or Limits

Parameter	Default	Maximum	Minimum
CHARACTER SET	WE8EBCDIC1047	N/A	N/A[1]
MAXDATAFILES	32	65534[2]	N/A
MAXINSTANCES	1	15	N/A
MAXLOGFILES	32	256	N/A
MAXLOGMEMBERS	2	5	N/A
MAXLOGHISTORY	100	65534	N/A

[1] N/A means that the platform-specific value is the same as the generic value.

[2] Specifying MAXDATAFILES larger than 40,000 requires using a value of 8K or larger for the DB_BLOCK_SIZE initialization parameter.

CREATE DATABASE

The parameters for the CREATE DATABASE SQL statement are listed in the following table:

Table B–3 CREATE DATABASE Parameters with z/OS-Specific Defaults or Limits

Parameter	Default	Maximum	Minimum
CHARACTER SET	WE8EBCDIC1047	N/A	N/A[1]
DATAFILE	Refer to "Oracle Database Files" in Chapter 4, "Defining z/OS Data Sets for the Oracle Database".	N/A	16K or one VSAM control area, whichever is larger
LOGFILE	Refer to "Oracle Database Files" in Chapter 4, "Defining z/OS Data Sets for the Oracle Database".	N/A	N/A
MAXDATAFILES	32	65534[2]	N/A
MAXINSTANCES	1	15	N/A
MAXLOGFILES	32	256	2
MAXLOGMEMBERS	2	5	N/A
MAXLOGHISTORY	N/A	N/A	N/A

[1] N/A means that the platform-specific value is the same as the generic value.

[2] Specifying MAXDATAFILES larger than 40,000 requires using a value of 8K or larger for the DB_BLOCK_SIZE initialization parameter.

Storage Parameters

The maximum values for DATAFILE, MINEXTENTS, NEXT, OPTIMAL, and PCTINCREASE are all limited by file size, which is limited by the size of the disk device.

C

Oracle Database for z/OS System Symbols

This appendix documents Oracle Database for z/OS system symbols for use in generating unique filenames.

System Symbols

Oracle Database for z/OS system symbols can be incorporated into SQL scripts and parameter files to guarantee that unique filenames are generated with values specific to the server instance or session.

Symbol	Description
&ORAPREFD	A high level (leftmost) dsname qualifier for use in database filenames. It is derived from the database region parameter DSN_PREFIX_DB or ORAPREFD For more information, refer to the description in chapter 3.
&ORASESST	A distinct session identifier for use in data set names. It is based on the OSDI pid (process id) in hexadecimal format. However, if the hexadecimal representation of the pid begins with 0-9, it is converted to G-P, respectively.
&ORASRVN	The OSDI service name without trailing blanks.

D National Language Support

This appendix documents the National Language Support (NLS) information specific to Oracle Database for z/OS. Information about the product-specific operation of language-specific features is provided in *Oracle Database Globalization Support Guide*.

The topics in this appendix include:

Overview

Oracle Database Globalization Support enables you to store, process, and retrieve data in native languages. National Language Support (NLS) is a subset of globalization support. It enables Oracle applications to interact with users in their native language, using their specific cultural conventions for displaying data.

The Oracle NLS architecture is data-driven, enabling support for specific languages and character encoding schemes to be added without requiring any changes in source code.

Supported Languages

Oracle Database for z/OS currently supports 24 languages. This table lists the languages that are supported and the default territories for each.

Language	Default Territory
American	America
Arabic	United Arab Emirates
Bengali	Bangladesh
Brazilian Portuguese	Brazil
Bulgarian	Bulgaria
Canadian French	Canada
Catalan	Catalonia
Croatian	Croatia
Czech	Czechoslovakia
Danish	Denmark
Dutch	Netherlands
Egyptian	Egypt
English	United Kingdom
Estonian	Estonia
Finnish	Finland
French	France
German	Germany
German DIN	Germany
Greek	Greece
Hebrew	Israel
Hungarian	Hungary
Icelandic	Iceland
Indonesian	Indonesia
Italian	Italy

Language	Default Territory
Japanese	Japan
Korean	Korea
Latin American Spanish	America
Latvian	Latvia
Lithuanian	Lithuania
Malay	Malaysia
Mexican Spanish	Mexico
Norwegian	Norway
Polish	Poland
Portuguese	Portugal
Romanian	Romania
Russian	CIS
Simplified Chinese	China
Slovak	Czechoslovakia
Slovenian	Slovenia
Spanish	Spain
Swedish	Sweden
Thai	Thailand
Traditional Chinese	Taiwan
Turkish	Turkey
Ukranian	Ukraine
Vietnamese	Vietnam

Overview of Character Set Support

Oracle automatically converts these types of data as they transfer between client and server, if required:

1. CHAR, VARCHAR, and LONG database columns
2. SQL and PL/SQL statements

3. host variables containing character data

Both the client and the server have associated character sets. The client declares its character set before connecting to the server through the NLS_LANG environment variable. On z/OS, this parameter is in the ORA$ENV DD statement.

If NLS_LANG is not specified, the default character set is assigned.

The character set for the server is declared when a database is created in Oracle and it cannot be changed once established. The default database character set is assigned if one is not explicitly declared.

When the client character set matches the server character set, character data is sent between client and server without any conversion. If the two character sets differ, all character data is converted from one character set to the other as it is transferred. It is important to be aware that all data contained in a database, whether in user-specified tables or in Oracle-specified data dictionary tables, is stored in the database character set.

Server-Side NLS

The character set in which the data is stored in the Oracle database is specified in the CHARACTER SET clause of the CREATE DATABASE statement. Refer to the *Oracle Database Administrator's Guide* for more information about the CREATE DATABASE statement.

When creating a database, the character set you choose depends on the language(s) to be supported and the character set(s) of the clients connecting to the server. Many languages can be supported by the default z/OS character set of WE8EBCDIC1047, but many others cannot.

If your site requires Euro support then you need to consider moving your server code page to WE8EBCDIC924. This code page is the EBCDIC version of ISO 8859-15.

The following table allows you to select an appropriate value for the CHARACTER SET clause of the CREATE DATABASE statement, based on the language to be supported and on the character set of the z/OS client. If you need to support languages that are not on this chart, please contact Oracle Support Services.

Language	Client Character Set on z/OS	Server Character Set on z/OS
Arabic	AR8EBCDICX	AR8EBCDIC420S

Language	Client Character Set on z/OS	Server Character Set on z/OS
Baltic	BLT8EBCDIC1112	BLT8EBCDIC1112S
Cyrillic	CL8EBCDIC1025	CL8EBCDIC1025R[1]
Eastern European	EE8EBCDIC870	EE8EBCDIC870S
Greek	EL8EBCDIC875	EL8EBCDIC875R[1]
Hebrew	IW8EBCDIC424	IW8EBCDIC424S
Icelandic	WE8EBCDIC871	WE8EBCDIC871S
Japanese	JA16DBCS	JA16DBCS
Korean	KO16DBCS	KO16DBCS
Simplified Chinese	ZHS16DBCS	ZHS16DBCS
Thai	TH8TISEBCDIC	TH8TISEBCDICS
Traditional Chinese	ZHT16DBCS	ZHT16DBCS
Turkish	TR8EBCDIC1026	TR8EBCDIC1026S
Western European	WE8EBCDIC924	WE8EBCDIC924
Western European	WE8EBCDIC1047	WE8EBCDIC1047
Western European	WE8EBCDIC1047E	WE8EBCDIC1047E

[1] Server character sets CL8EBCDIC1025S (Cyrillic) and EL8EBCDIC875S (Greek) on z/OS have some errors. Oracle Corporation recommends converting from those character sets to CL8EBCDIC1025R and EL8EBCDIC875R, respectively.

Client-Side NLS

The client character set is determined by the data that is sent to the Oracle server. This is typically determined by the type of terminal that is used by the client. See your system administrator for more information about the character set used by your terminal.

Supported z/OS Client Character Sets

The default client character set has changed from WE8EBCDIC37C to WE8EBCDIC1047, but most existing Oracle7 for OS/390 databases are WE8EBCDIC37C. When the client and server character sets differ, the CPU overhead increases. If you set NLS_LANG so that the client and server character sets match, the CPU overhead decreases.

The following character sets are supported for z/OS clients:

Character Set	Support
AR8EBCDICX	XBASIC Code Page 420 Arabic
BLT8EBCDIC1112	EBCDIC Code Page 1112 Baltic/Multilingual
CL8EBCDIC1025	EBCDIC Code Page 1025 Cyrillic/Multilingual
D8EBCDIC273	EBCDIC Code Page 273 Austrian /German
DK8EBCDIC277	EBCDIC Code Page 277 Danish
EE8EBCDIC870	EBCDIC Code Page 870 East European
EL8EBCDIC875	EBCDIC Code Page 875 Greek
F8EBCDIC297	EBCDIC Code Page 297 French
I8EBCDIC280	EBCDIC Code Page 280 Italian
IW8EBCDIC424	EBCDIC Code Page 424 Hebrew
JA16DBCS	EBCDIC DBCS Japanese
KO16DBCS	EBCDIC DBCS Korean
S8EBCDIC278	EBCDIC Code Page 278 Swedish
TH8TISEBCDIC	EBCDIC Code Page 838 Thai
TR8EBCDIC1026	EBCDIC Code Page ISO 8859 Turkish
WE8EBCDIC37	EBCDIC Code Page 37 West European
WE8EBCDIC37C	EBCDIC Code Page 37 West European with extensions
WE8EBCDIC284	EBCDIC Code Page 284 Spanish (Spain)
WE8EBCDIC285	EBCDIC Code Page 285 English, UK
WE8EBCDIC500	EBCDIC Code Page 500 West European
WE8EBCDIC871	EBCDIC Code Page 871 Icelandic
WE8EBCDIC1047	EBCDIC Code Page 1047 Latin 1/Open Systems
WE8EBCDIC1140	EBCDIC code page 1140 Western European with Euro support

Character Set	Support
WE8EBCDIC1140C	EBCDIC code page 1140 Western European with extensions and Euro support
ZHS16DBCS	EBCDIC DBCS Simplified Chinese
ZHT16DBCS	EBCDIC DBCS Traditional Chinese

NLS Calendar Utility (LXEGEN)

The LXEGEN utility is used to customize calendar data, as described in *Oracle Database Globalization Support Guide*. Customizing calendar data is done primarily through z/OS UNIX System Services. You will need to be familiar with z/OS UNIX System Services in general, as well as the material in *Oracle Database User's Guide for IBM z/OS* describing the particular environment variables required by Oracle products in a z/OS UNIX System Services shell environment.

The z/OS implementation of Oracle products uses the same filenames and directory structure that the Oracle product documentation uses. Log on to z/OS UNIX System Services and verify that you can locate the applicable directories and files. If you have any difficulty locating these files, ensure that the ORACLE_HOME environment value is properly set and that z/OS UNIX System Services is properly installed before attempting to customize yor calendar data.

Index