

Oracle® Database

User's Guide

10g Release 2 (10.2.0.5) for IBM z/OS on System z

B25396-02

October 2012

Oracle Database User's Guide, 10g Release 2 (10.2.0.5) for IBM z/OS on System z

B25396-02

Copyright © 2006, 2012, Oracle and/or its affiliates. All rights reserved.

Primary Author: Bharathi Jayathirtha

Contributing Author: Server Technologies Porting

Contributor: Janelle Simmons

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, delivered to U.S. Government end users are "commercial computer software" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, shall be subject to license terms and license restrictions applicable to the programs. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information on content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services.

Contents

Preface	ix
Audience	ix
Documentation Accessibility	ix
Command Syntax	ix
Accessing Documentation	x
Related Documentation	xi
Typographic Conventions	xi
1 Overview of Oracle Database Products	
Servers	1-1
Tools and Utilities	1-2
Networking	1-2
Application Development	1-3
Access Managers	1-3
Oracle Call Interface (OCI) and Oracle C++ Call Interface (OCCI)	1-4
Other Software	1-4
2 Oracle Software Interaction with z/OS and Language Environment	
Overview	2-1
Execution Environments on z/OS	2-2
POSIX ON vs. POSIX Shell Environments	2-2
Runtime Program Access Requirements	2-2
Application Program, Tool, or Utility	2-3
LE Runtime	2-3
Oracle Program Interface Code	2-4
Oracle Supporting Data Objects	2-4
Invocation Examples	2-5
Tool and Utility Parameter Conventions	2-6
Parameters in Batch (JCL)	2-6
Parameters in TSO	2-7
Parameters in a z/OS UNIX Shell	2-8
Parameters Containing Spaces	2-9
Environment Variables	2-9
Environment Variables in a z/OS UNIX Shell	2-10
Environment Variables in Batch and TSO Environments	2-10

Oracle Environment Variable Files	2-11
Setting Environment Variables in a Program	2-12
Files, File Name Syntax, and File Processing	2-13
Filespecs.....	2-13
File Types and Filespec Syntax	2-14
Data Set Names	2-14
DD Names.....	2-15
SYSOUT	2-16
TSO Terminal.....	2-17
HFS Files.....	2-18
Standard Files and Redirection	2-19
Standard File Defaults (non-POSIX)	2-19
Redirection Operators in TSO and Batch	2-20
File Name Manipulation	2-21
What's in a Name?	2-21
Data Set Name Parts	2-22
HFS File Name Parts.....	2-22
Extension Processing	2-23
FNA.....	2-24
FNA Name Transformation	2-24
FNA Attribute Assignment	2-25
Example FNA Control File	2-26
Data Set DCB Attributes.....	2-27
Default DCB Attributes.....	2-27
DCB Attributes from FNA.....	2-27
DCB Attributes without FNA	2-28
Interrupt Processing in TSO and z/OS UNIX Shell	2-28
Oracle C Runtime Compatibility	2-29
Controlling Compatibility.....	2-29
Filespec Compatibility.....	2-29
File Attribute Compatibility	2-30
Data Set Name Prefixing Compatibility	2-30
Standard Files and Redirection Compatibility	2-31

3 Oracle Net and Server Connections on z/OS

Protocols	3-1
Protocol and Address Determination	3-2
XM Protocol	3-3
XM Protocol Address	3-4
XM Protocol Examples	3-4
TCP Protocol	3-5
Oracle Net Files	3-6
Profile Parameters (sqlnet.ora).....	3-6
LOG_DIRECTORY_CLIENT.....	3-7
LOG_FILE_CLIENT.....	3-7
TRACE_DIRECTORY_CLIENT	3-7
TRACE_FILE_CLIENT.....	3-7

TNSPING.TRACE_DIRECTORY.....	3-7
Local Naming Parameters (tnsnames.ora)	3-8
Directory Usage Parameters (ldap.ora).....	3-8
Oracle Net Output Files	3-8

4 Oracle Tools and Utilities on z/OS

SQL*Plus	4-1
Invocation.....	4-1
Output Interruption.....	4-2
Profile Files.....	4-2
SQL Files.....	4-3
EDIT Command Processing	4-3
SPOOL Command Processing.....	4-4
HOST Command Processing.....	4-4
TIMING Processing	4-6
Return Codes	4-6
Unsupported Features.....	4-6
SQL*Plus Examples.....	4-7
SQL*Loader	4-8
Invocation.....	4-8
SQL*Loader Files and Filespecs	4-9
Control File	4-9
Log File	4-10
Data Files.....	4-10
Bad and Discard Files	4-11
VSAM Data File Processing Considerations	4-12
Bad and Discard File Attributes.....	4-12
SQL*Loader Return Codes.....	4-13
SQL*Loader Examples.....	4-13
Export and Import	4-14
Invocation.....	4-14
Export File	4-15
Cross-System Export/Import.....	4-16
Export and Import Return Codes	4-16
Unsupported Features.....	4-16
Export and Import Examples	4-16
Datapump Export and Import	4-17
Invocation.....	4-17
Datapump Export and Log Files.....	4-18
Datapump Processing in the Server	4-19
Datapump Export and Import Interactive Mode	4-19
Datapump Export and Import Return Codes	4-19
TKPROF	4-19
Invocation.....	4-19
Input Trace File.....	4-20
Output Files.....	4-20
Return Codes	4-21

TKPROF Example	4-21
----------------------	------

5 Oracle Server Considerations on z/OS

Character Data	5-1
Character Data and SQL Queries.....	5-1
Character Data and Partitioned Tables	5-2
Characters in SQL Statements	5-2
File Processing in the Server	5-3
Security Considerations	5-3
Database Directory Objects.....	5-4
UTL_FILE	5-4
UTL_FILE Example	5-5
JAVA File I/O.....	5-6
External LOBs (BFILEs).....	5-6
External Tables	5-6
Datapump Export and Import	5-7

6 Developing Oracle Applications to Run on z/OS

Overview	6-1
Application Design Considerations	6-3
Basic Application Requirements	6-4
Requirements for Complex Applications	6-5
Using z/OS Assembler Language.....	6-5
Dynamic Linkage Techniques.....	6-5
Using DLLs.....	6-5
Using the XPLINK(ON) LE Runtime Option	6-5
Using COBOL Dynamic Linkage	6-6
Using C/C++ system()	6-6
Using C/C++ fetch()	6-6
Using z/OS LOAD/CALL or LINK	6-6
Using z/OS ATTACH.....	6-7
Multiple LE Enclaves in an Application.....	6-7
Multitasking Applications.....	6-7
z/OS Environment and z/Architecture Hardware States.....	6-8
POSIX Threading	6-8
OCI Interface to Publish/Subscribe	6-8
Building an Application	6-9
Precompiling Programs.....	6-9
Oracle Precompiler Executables	6-9
Oracle Precompiler INCLUDE Files	6-10
Oracle Precompiler Options	6-10
INAME	6-11
ONAME	6-11
LNAME	6-11
Configuration Files	6-11
Return Codes	6-12
Language-Specific Coding Considerations	6-12

Compiler Support Considerations	6-12
Pro*COBOL Considerations.....	6-12
Pro*C/C++, OCI, and OCCI Considerations	6-13
Pro*FORTRAN Considerations	6-13
Pro*PL/I Considerations	6-13
Special Considerations for Running Precompilers in Batch and TSO Environments ...	6-13
Sample JCL for Running the Pro*C/C++ Precompiler in Batch Environments.....	6-13
Sample JCL for Running the Pro*COBOL Precompiler in Batch Environments.....	6-14
Sample JCL for Running the Pro*FORTRAN Precompiler in Batch Environments	6-14
Sample JCL for Running the Pro*PL/I Precompiler in Batch Environments	6-15
Sample Commands for Running Oracle Precompilers in a Shell	6-15
Compiler Options for Oracle Applications	6-15
C/C++ Compiler Options.....	6-16
COBOL Compiler Options	6-16
PL/I Compiler Options.....	6-17
Compiling Programs	6-17
Sample Commands for Compiling Programs in a Shell	6-17
Compiling OCCI programs to Run in Batch and TSO Environments	6-18
Linking Programs.....	6-18
Linking Pro*C/C++, OCI, COBOL, and PL/I Programs to Run in Batch and TSO Environments	6-18
Linking COBOL Programs Using DYNAM.....	6-19
Linking FORTRAN programs to Run in Batch and TSO Environments	6-19
Linking API Stub Programs to Run in a Shell	6-20
Sample Commands for Compiling and Linking API Stub Programs in a Shell.....	6-20
Linking DLL Stub Programs in a Shell	6-21
Sample Commands for Compiling and Linking DLL Stub Programs in a Shell.....	6-22
Binding OCCI programs to Run in Batch and TSO Environments	6-22
Building an Oracle XDK API Using Program in a Shell	6-23
Using Oracle-Supplied Procedures to Build Applications	6-23
Sample Batch JCL to Build an Oracle Precompiler Program.....	6-24
Using Make to Build a Precompiler Program.....	6-24
Pro*COBOL Sample Programs	6-24
Pro*FORTRAN Sample Programs.....	6-24
Pro*C/C++ Sample Programs	6-25
Batch JCL to Build OCI Programs	6-25
Batch JCL to Build OCCI Programs.....	6-25
Using Make to Build OCI and OCCI Programs	6-25
Sample OCI and OCCI Programs.....	6-25
Building a Traditional Load Module with the Alternate API Stub	6-26
Method 1: Prelink and Link	6-26
Method 2: Precompile and/or Compile with Name Mapping	6-27
Method 3: Link.....	6-29
Application Runtime Considerations	6-29
Oracle Interface Initialization, Processing, and Error Handling.....	6-30
Application Resources and Cleanup	6-32
Developing Oracle Applications for the CICS TS Environment.....	6-33

CICS TS Application Design Considerations.....	6-33
CONNECT Statements.....	6-33
Synchronization of Oracle and CICS TS Updates	6-34
Cursor Considerations	6-34
Accessing Multiple Oracle Databases.....	6-34
Accessing Oracle Database 10g and DB2 Databases in a Single Transaction.....	6-34
Additional SQL Statement Restrictions	6-35
CICS TS EDF and Oracle SQL Statements	6-35
Environment Variables.....	6-35
Considerations for Building a CICS TS application	6-35
Running CICS TS Programs	6-35
Developing Oracle Applications for the IMS TM Environment.....	6-35
IMS TM Application Design Considerations.....	6-36
IMS TM Versions Supported.....	6-36
CONNECT Not Supported.....	6-36
Synchronization of Oracle and IMS TM Updates	6-36
Cursor Considerations	6-37
Accessing Multiple Oracle Databases.....	6-37
Additional SQL Statement Restrictions.....	6-37
Accessing Oracle Database 10g and DB2 Databases in a Single Transaction.....	6-37
Processing of Oracle Database 10g Errors by Your IMS TM Program.....	6-37
Environment Variables.....	6-38
Considerations for Building an IMS TM application.....	6-38
Running IMS TM Programs.....	6-38

7 Migration from Earlier Oracle Versions

Overview.....	7-1
Migrating from Oracle8i.....	7-1
Migration Checklist.....	7-2
Migrating from Oracle9i.....	7-3
Normalized File Access.....	7-3
Global Environment File	7-4
Use of LE Exit CEEBXITA.....	7-4
Oracle Runtime Compatibility	7-4
SYSOUT Filespec in Clients.....	7-4
SQL*Loader Changes.....	7-4
Interpretation of DDN Keywords	7-5
Default Filespecs for DD-type Data Files	7-5
Default Filespecs for Data Set Name Files.....	7-5
Default Bad/Discard DCB attributes.....	7-5

A Environment Variables Used by Oracle Database for z/OS

Environment Variables Used by Oracle Database.....	A-1
--	-----

Index

Preface

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

Audience

This guide is intended for those who are responsible for performing tasks such as:

- Running Oracle Database tools or utilities, such as SQL*Plus, Export, Import, or SQL*Loader on z/OS
- Designing or developing Oracle Database applications using Oracle Precompilers or the Oracle Call Interface (OCI) on z/OS
- Running Oracle Database application programs on z/OS

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at <http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc>.

Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit <http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info> or visit <http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs> if you are hearing impaired.

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/rdskc0t1d0s6 of=/dev/rst0 bs=10b count=10000
braces { }	Braces indicate required items: .DEFINE {macro1}
brackets []	Brackets indicate optional items: cvtcrct <i>termname</i> [<i>outfile</i>]
ellipses ...	Ellipses indicate an arbitrary number of similar items: CHKVAL <i>fieldname</i> <i>value1</i> <i>value2</i> ... <i>valueN</i>
<i>italics</i>	Italic type indicates a variable. Substitute a value for the variable: <i>library_name</i>
vertical line	A vertical line indicates a choice within braces or brackets: FILE <i>filesize</i> [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 10g products is available in both HTML and PDF formats in the following locations:

- In the `doc` subdirectory on the Oracle Database 10g 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 10g Documentation Library link.
- Online on the Oracle Technology Network (OTN) Web site:
<http://www.oracle.com/technetwork/indexes/documentation/index.html>

Related Documentation

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

- Oracle Database
 - *Oracle Database Release Notes for IBM z/OS on System z*
 - *Oracle Database Installation Guide for IBM z/OS on System z*
 - *Oracle Database User's Guide for IBM z/OS on System z*
 - *Oracle Database Messages Guide for IBM z/OS on System z*
 - *Oracle Database System Administration Guide for IBM z/OS on System z*

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

<http://www.oracle.com/technetwork/indexes/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.
<i>italic</i>	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.

Overview of Oracle Database Products

Oracle delivers a number of software products and facilities on the z/OS platform, including its core relational database server, related tools and utilities, networking features, 3GL application development and deployment facilities, and integration products unique to the platform. This set of products and facilities allows z/OS systems to host the same Oracle platform as UNIX, Linux, and other systems on which the Oracle database runs. The integration products allow widely used z/OS components such as CICS TS, IMS TM, and DB2 to participate in an Oracle Database application. What follows are overviews of the Oracle product set on z/OS organized by functional category.

This chapter contains the following sections:

- [Servers](#)
- [Tools and Utilities](#)
- [Networking](#)
- [Application Development](#)
- [Other Software](#)

Servers

Foremost in this category is the Oracle relational database server, which has been ported to z/OS, OS/390, and predecessor IBM MVS operating systems since Oracle RDBMS Version 5 (1986). The current product is Oracle Database 10g release 2 (10.2) for IBM z/OS on System z. As with all prior Oracle versions, the z/OS implementation of this product is compiled from the same C language source code as on other platforms, differing only in the thin layer of programming that adapts it to the host operating system. This means the SQL and PL/SQL languages, Java facilities, SQL statement optimizer, and other Oracle features work the same on z/OS as they do on other Oracle platforms. Oracle Database applications developed with a non-z/OS Oracle server generally can be switched to a z/OS Oracle server or vice versa without changes.

Despite this commonality, the Oracle Database for z/OS makes extensive use of features unique to the operating system: it is managed by a formal z/OS subsystem, uses cross-memory services for inter-address-space operations, and exploits z/OS Workload Manager (WLM) facilities to classify and dispatch database requests from remote clients. Because of current limitations in IBM's Language Environment (LE) for z/OS support for 64-bit virtual memory addressing, a patented multi-address-space server architecture allows Oracle Database for z/OS to support far larger workloads than the available 31-bit addressing would normally permit. The Oracle software

component that provides the subsystem and address space management facilities is called OSDI (Operating System Dependent Interface) and is unique to z/OS.

In addition to its relational database server, Oracle offers Transparent Gateways on z/OS so that non-Oracle data sources can participate in Oracle database applications. A Transparent Gateway allows an Oracle database server to access non-Oracle data through Oracle's distributed database mechanism. Two Transparent Gateway products are available to run on z/OS: the Transparent Gateway for DB2 and the Transparent Gateway for iWay. The latter product uses the Information Builders iWay product to provide access to a variety of z/OS data sources including Adabas, VSAM, IMS/DB and others. Like the database server, the Transparent Gateways on z/OS run under control of OSDI.

Access to an Oracle Transparent Gateway is always through an Oracle database server; client applications do not connect directly to a Transparent Gateway. Both local database servers (on the same z/OS system) and remote database servers (on any Oracle platform) can access a Transparent Gateway on z/OS.

Your installation may or may not run Oracle database or Transparent Gateway servers on z/OS. Some customers choose to install only Oracle client components on z/OS (tools and utilities and application development facilities) to enable access to Oracle servers on other platforms. This is referred to as a client-only install in later sections.

Tools and Utilities

Tool and utility programs are associated with the Oracle database server for various purposes. These include Oracle Export and Import for transporting database objects, SQL*Plus for ad hoc SQL and reporting, SQL*Loader for high-speed database table loading, and others. Like the Oracle database server, the z/OS versions of these tools and utilities are compiled from the same common code base and function in the same way as their counterparts on other Oracle platforms.

On z/OS, most Oracle tools and utilities can run in traditional MVS environments (batch job and native TSO) and the z/OS POSIX environments such as a Telnet z/OS UNIX System Services shell session or the OMVS shell in TSO. In the traditional environments they behave like typical MVS utility programs including the use of DD and data set names for input and output files and support for TSO Command Processor (CP) invocation. In z/OS POSIX environments, they exhibit UNIX-like behavior and typically access Hierarchical File System (HFS) files. However, in both types of environments, the same tool or utility executable (program object) is used.

None of the Oracle tools or utilities is supported under CICS TS or IMS TM. In those environments, only user-written 3GL applications are supported as discussed in [Chapter 6, "Developing Oracle Applications to Run on z/OS."](#)

Although associated with the Oracle database server, Oracle tools and utilities are also provided in a client-only install on z/OS. As discussed in the following section, Oracle tools and utilities can access a local (z/OS) or remote (any platform) Oracle database server with equal ease. For example, you can run SQL*Loader on z/OS and have it load data from a z/OS VSAM data set into an Oracle database server running on UNIX or Linux.

Networking

Oracle Database for z/OS views all client connections to the database as network connections, even when the client and server are on the same system and no real network is involved. This means virtually all Oracle client/server interactions work the same way whether the client is local or remote, and neither client applications nor

the server being accessed are aware of the distinction. Applications can be developed and tested with a local server and then deployed, unchanged, to a production environment with a remote server (or vice versa). Similarly, a production database server or a production client environment can be moved from one system to another with minimal disturbance.

The networking feature, called Oracle Net, is built into the Oracle server and the client-side Oracle program interface code. The same client interface code supports both Oracle tools and utilities and user-written Oracle applications, so both tools and utilities and your own programs have the same mobility in terms of server access.

On z/OS, local Oracle connections (between a client application and a server on the same z/OS image) use z/OS cross-memory services. Besides being efficient, the cross-memory connection protocol has the characteristic that the application processing in the server is executed by the application task. Server requests run at the application's Workload Manager goal and the processing time consumed by the requests is charged to the application, not to the Oracle database server.

For real networking, Oracle Net on z/OS uses the TCP/IP protocol. The terms inbound and outbound refer to two different network situations as viewed from z/OS. An inbound client means a client on a remote system connecting to an Oracle database server on a local z/OS system. A client application that is running on a local z/OS system and connecting to a remote Oracle database server is an outbound client. In either case, the remote client or remote server can be on another z/OS system or on any other Oracle platform. When client and server are on dissimilar platforms, translation of both character and number data are handled automatically.

Supporting inbound clients on z/OS requires an Oracle Net Services listener. The listener provides the network endpoint (TCP/IP port) to which remote clients initially connect, and takes care of routing the client to the correct Oracle database server.

Outbound clients on z/OS do not use Oracle Net Services; they interact directly with the TCP/IP implementation provided by the system. However, they do interact with a listener on the target remote system.

Application Development

Support for 3GL application development on z/OS is provided by Oracle Precompilers: Pro*C/C++, Pro*COBOL, Pro*FORTRAN, and Pro*PL/I. These products read source code containing imbedded EXEC SQL directives and translate the directives into appropriate data declarations, executable statements, and Oracle API calls to access an Oracle database server. Output from the precompiler is input to a supported IBM language compiler and ultimately to the z/OS binder (linkage editor) to produce an executable load module or program object. Precompiler applications can run in a native z/OS batch or TSO address space, in CICS TS and IMS TM transaction environments, or in a z/OS POSIX environment such as a TSO OMVS shell or rlogin shell.

Access Managers

Application execution under CICS TS or IMS TM requires the use of an Oracle integration product called an Access Manager. The Access Managers provide coordination between Oracle database updates and commit or rollback processing in the respective transaction manager. This enables you to write transaction programs that update both Oracle resources and non-Oracle resources, such as VSAM or IMS/DB data, with full two-phase commit integrity. Access Managers must be

installed and configured with their respective IBM transaction manager before Oracle-accessing transactions are run.

Oracle Call Interface (OCI) and Oracle C++ Call Interface (OCCI)

In addition to the precompiler products, APIs called Oracle Call Interface (OCI) and Oracle C++ Call Interface (OCCI) can be used to develop 3GL Oracle applications on z/OS. With OCI and OCCI, you write a C (OCI only) or C++ language program that makes calls to specific functions to perform specific operations in the Oracle server. While more complex to use than the precompilers, OCI and OCCI provide access to all features of the Oracle database server. On z/OS, OCI and OCCI applications can be built to run in batch and TSO or as z/OS POSIX programs. A few OCI features use POSIX threading and so require a POSIX environment. OCI and OCCI applications are not supported in CICS TS or IMS TM.

Other Software

Other Oracle components are provided on z/OS but they are less likely to be encountered by end users and application developers. In this category are management infrastructure components such as the Enterprise Manager Management Agent and database administration tools such as Recovery Manager. For more information, refer to the *Oracle Database System Administration Guide for IBM z/OS on System z*.

Oracle Software Interaction with z/OS and Language Environment

This chapter describes how Oracle software on z/OS is integrated with the operating system. It discusses the common implementation attributes of Oracle components used by end users and application developers. Details on specific Oracle components are provided in later chapters.

This chapter contains the following sections:

- [Overview](#)
- [Execution Environments on z/OS](#)
- [Files, File Name Syntax, and File Processing](#)
- [Interrupt Processing in TSO and z/OS UNIX Shell](#)
- [Oracle C Runtime Compatibility](#)

Overview

Except for components written in PL/SQL or Java, Oracle software that runs on z/OS is written in the C or C++ language. Oracle uses the IBM C/C++ compiler to build its z/OS components and Language Environment (LE) for z/OS to provide some of the operating system runtime services required during program execution. This means C/C++ and LE conventions figure heavily in the behavior of Oracle software on z/OS. For example, LE-specific syntax is normally used when specifying data set or HFS file names to an Oracle tool or utility.

Oracle on MVS predates both IBM's C/C++ compiler and LE by a number of years. In the early 1980's, Oracle developed and used its own C compiler and runtime library to support its products on MVS. Certain features of Oracle's runtime library continue to be supported on z/OS so that existing customer JCL and scripts continue to work unmodified. This legacy support causes slight deviations from normal LE behavior in several areas; it is optional and can be disabled if desired.

Besides the optional Oracle runtime compatibility feature, there are a number of areas where LE does not provide services that Oracle software requires, and those needs are met by using Oracle-developed program code. In some cases these services are implemented as extensions to existing LE support. For example, LE does not provide file syntax for directly requesting SYSOUT (JES spool) output. Oracle provides that capability, using file name syntax that is a logical extension of existing LE syntax. Other non-LE program support in Oracle, such as TSO attention interrupt support, is completely separate from LE.

Execution Environments on z/OS

Oracle supports several different z/OS environments for executing Oracle tools and utilities and user-written Oracle applications. Both LE and certain parts of Oracle program code behave differently across these environments. Applications that run under CICS TS or IMS TM are controlled by Oracle Access Manager software and are subject to the behaviors and requirements of the specific Access Manager. For more information, refer to [Chapter 6, "Developing Oracle Applications to Run on z/OS."](#) The remaining environments are discussed here, and are as follows:

- Batch, meaning an environment started from JCL that is neither a TSO session nor a z/OS UNIX shell. This includes normal batch jobs that run in an initiator and started task (STC) address spaces that do not use an initiator.
- TSO, meaning a TSO/E logon session that is not running a z/OS UNIX shell. This does not include batch TMP jobs, which for most purposes are treated as a batch environment by Oracle.
- z/OS UNIX shell, meaning a Telnet, rlogin, or TSO logon session running a z/OS UNIX System Services shell.

z/OS provides a set of POSIX-compliant UNIX services. Originally called OpenEdition MVS, they are now called z/OS UNIX System Services. The most common way to use these services is from the z/OS UNIX shell variations described in the previous list, which provide a UNIX-style shell environment with a Hierarchical File System (HFS) and implementations of widely-used UNIX utilities like `awk`, `grep`, and `vi`.

POSIX ON vs. POSIX Shell Environments

Any task in a z/OS address space can interact with z/OS UNIX System Services provided it is authorized to do so. For example, an application program does not have to be launched from a z/OS UNIX shell to use z/OS UNIX System Services features such as HFS files or POSIX threading. Conversely, programs launched from a z/OS UNIX shell generally are free to use z/OS services and facilities that do not conflict with the mechanisms of z/OS UNIX Systems Services.

Many of the significant behavior differences in application programs are associated with LE, not with being in a z/OS UNIX shell. LE has its own internal POSIX indicator that is set to either ON or OFF when a program begins executing, and remains so for the life of the program (technically, for the life of the LE enclave). The initial setting of this POSIX indicator can come from the application program, from an external runtime specification, or can be defaulted based on the environment. In the case of Oracle tools and utilities, POSIX is set to ON if the program executable was obtained from HFS (for example, as a shell command). Otherwise, it is set to OFF (for example, with conventional batch and TSO execution).

Thus, the phrase "in a POSIX environment" means when a program is executing with POSIX set to ON, regardless of whether the program executes in a z/OS UNIX shell or in an ordinary batch job or TSO session. For behaviors that depend specifically on a z/OS UNIX System Services shell, the phrase "in a z/OS UNIX shell" or a reference to the specific shell will be used.

Runtime Program Access Requirements

Running a program that accesses an Oracle database on z/OS, including Oracle tools and utilities and Pro*C/C++, Pro*COBOL, Pro*FORTRAN, Pro*PL/I, OCI, or OCCI applications, requires the following components to be accessible:

- Application program, tool, or utility

- LE runtime
- Oracle program interface code
- Oracle supporting data objects

How these components are accessed depends on the execution environment and on how Oracle and LE software are installed and configured on your system. This, in turn, may dictate how you code the JCL, what things you ALLOCATE in a TSO session, or the environment variables you set in a z/OS UNIX shell. This section explains how these components are accessed in general. You may need to ask your z/OS system administrator or Oracle database administrator for the installation specifics of your system.

Application Program, Tool, or Utility

Oracle tools and utilities usually are installed in a specific load library data set, referred to in this document as the CMDLOAD data set. The exact name of the data set is chosen by the installation and can vary over different systems and over different versions or releases of Oracle software. For environments other than a z/OS UNIX shell, you may need to specify this data set in JCL (in a JOBLIB or STEPLIB DD statement) or in a TSO command such as CALL or TSOLIB. However, if your installation adds the CMDLOAD data set to the system linklist or copies the CMDLOAD modules into an existing linklist or LPA library, the modules are available in all environments without specifying a data set.

For z/OS UNIX shell environments, Oracle tools and utilities normally reside in HFS, in the `bin` subdirectory of the installation Oracle home directory. Like the name of the CMDLOAD data set, the exact path for the Oracle home directory is chosen by the installation. This document refers to this directory as the Oracle home directory or as `ORACLE_HOME`, which is the name of an environment variable normally set to the Oracle home directory path. In the z/OS UNIX shell, as in other UNIX systems, the `PATH` environment variable is used to specify the directories in which to search for an executable. Thus, to access Oracle tools and utilities in a z/OS UNIX shell you would normally include the `bin` subdirectory of the Oracle home directory in your `PATH` environment variable.

Oracle application programs can reside in a normal load library data set, linklist or LPA (link pack area) data set, or z/OS UNIX System Services HFS directory. Follow the conventions or standards of your installation to make your application program available for execution.

LE Runtime

It is typical for LE runtime to be available in all supported z/OS Oracle execution environments without special action on your part. In the rare situation where it is not, you will need to specify LE runtime library data sets in your JOBLIB or STEPLIB concatenation or in a TSOLIB command in TSO. Check with your z/OS system administrator if you are unsure of LE availability or to determine the data set names for LE runtime services, if needed.

LE is required in all z/OS UNIX shells, so no special action is required to make LE available to a shell-based Oracle application.

It is possible for a single z/OS system to have multiple releases or maintenance levels of LE installed, and LE provides a mechanism called Run-Time Library Services (RTLS) to manage this. If your installation is using RTLS, consult with your z/OS system administrator to determine how to access the correct release of LE for your Oracle database application programs.

Oracle Program Interface Code

The Oracle program interface code is a program object named LIBCLNTS. It is required at runtime by all Oracle programs on z/OS including Oracle tools and utilities and any applications that access the Oracle database. In conventional batch and TSO environments, LIBCLNTS normally resides in the same CMDLOAD data set as the Oracle tools and utilities. If your installation has made CMDLOAD a linklist library, or if LIBCLNTS has been placed in a linklist or LPA library, the interface code can be accessed from any batch job or TSO session without special action. Otherwise, the CMDLOAD data set must be included in a JOBLIB or STEPLIB DD statement or, in TSO, it must be specified on the CALL or TSOLIB command.

Besides LIBCLNTS, if you are exploiting Oracle's limited support for the COBOL DYNAM compile option in a Pro*COBOL application, module ORASTBX and all of its aliases must be available in JOBLIB, STEPLIB, TSOLIB, or the system linklist. This module is the loadable form of the Oracle API stub used only by COBOL DYNAM applications. Normally it is installed in the same location as LIBCLNTS.

In TSO environments only, an additional runtime module named ORASTAX is used by Oracle database applications to process TSO Attention or PA1 key interrupts. Like LIBCLNTS, ORASTAX normally resides in the Oracle CMDLOAD data set but can be placed in a linklist or LPA library.

For applications running in a POSIX environment, LIBCLNTS is provided as an HFS DLL (Dynamic Link Library) file, normally in the `lib` subdirectory of Oracle home. Because of the way it is called (through the LE DLL mechanism), the LIBPATH environment variable determines the directories that are searched. LIBPATH is similar to PATH but pertains to DLL access rather than normal command access. When running an Oracle-accessing program in a POSIX environment, LIBPATH must include the directory that contains LIBCLNTS.

For OCCI applications, the OCCI program interface code is a program object named LIBOCCI. It is required at runtime by all OCCI applications. LIBOCCI and OCCI applications that call the OCI API also depend on LIBCLNTS. Therefore, LIBCLNTS and LIBOCCI reside in CMDLOAD and in the `lib` subdirectory of Oracle home.

Oracle Supporting Data Objects

Besides executable code, the Oracle program interface uses a set of data objects containing character set translation tables, the text of displayable messages in various languages, and time zone configuration data. These are associated with Oracle's National Language Support (NLS) features and they are required, even if an application uses only the default language and character set.

For z/OS, these objects are distributed in two forms: as non-executable program objects residing in a partitioned data set (PDS) and as z/OS UNIX System Services HFS files. The program object form is used in all environments, including POSIX environments, if possible. If the program object form is not available in a POSIX application, Oracle accesses the HFS form of these files. (A non-POSIX application does not use the HFS form. If the program object form is not available, then the non-POSIX application fails during Oracle initialization.)

The program object forms of the data and message objects normally are installed in a data set referred to as the MMSG data set. Like CMDLOAD, the exact name of this data set is chosen by the installation. Usually it has the same high-level qualifiers as the corresponding CMDLOAD. There are two ways to make the MMSG data set available to an application, as follows:

- Supply an ORA\$LIB DD statement specifying the MESH data set, or ALLOCATE a comparable ORA\$LIB DD to a TSO session. This is the recommended approach for applications that are not running in a z/OS UNIX shell.
- Include the MESH data set in your JOBLIB or STEPLIB concatenation, or in your TSOLIB command in TSO.

It also is possible for an installation to make the MESH objects globally available by placing them in a linklist or LPA library, but this is not recommended.

Oracle tools and utilities and the Oracle program interface code use the following sequence to locate and access NLS data and messages:

1. If an ORA\$LIB DD is present in the address space, it is searched for a matching program object.
2. If no ORA\$LIB DD is present (or if the attempt to load from ORA\$LIB fails), POSIX is ON, and the ORACLE_HOME environment variable is defined, Oracle attempts to read an HFS file in a subdirectory of ORACLE_HOME.
3. If neither of the previous steps occurred or succeeded, Oracle issues a z/OS LOAD request for the object. This LOAD follows the normal z/OS default search order for modules and program objects.

Because it can be awkward to include DD statements in POSIX applications, most such applications rely on the HFS form of these files (as describe previously, in step 2). Conversely, most batch and TSO applications rely on an ORA\$LIB DD statement.

Invocation Examples

This section illustrates invoking a program that accesses the Oracle database and providing access to the required loadable components discussed previously. They are not complete examples of usage. All examples assume that LE is globally available through the linklist or LPA.

Example 1

The following example shows part of the batch JCL used to invoke an Oracle Pro*COBOL application named ODBRA20. The application module resides in data set ODB1.PROD.LINKLIB (which is not a linklist library) and the Oracle program interface code also is not in the linklist or LPA, so the CMDLOAD data set also is included in STEPLIB. The CMDLOAD and MESH data sets have been installed with the prefix ORACLE.V10G:

```
//STEP07 EXEC PGM=ODBRA20
//STEPLIB DD DISP=SHR,DSN=ODB1.PROD.LINKLIB
//          DD DISP=SHR,DSN=ORACLE.V10G.CMDLOAD
//ORA$LIB DD DISP=SHR,DSN=ORACLE.V10G.MESH
```

Example 2

In the following example, a TSO session performs the necessary allocations to invoke Oracle SQL*Plus as a command processor. The installed data set conditions are the same as those in Example 1.

```
READY
tsolib activate da('oracle.v10g.cmdload')
READY
alloc shr fi(ora$lib) da('oracle.v10g.mesg')
READY
sqlplus
```

```
SQL*Plus: Release 10.2.0.5.0 - Production...
```

Example 3

The following example is also SQL*Plus, but in the POSIX environment in a Telnet TTY session. The HFS part of the Oracle install created `/fs1/oracle/prod` as the Oracle home directory. NLS data and message files from HFS are used.

```
$ export ORACLE_HOME=/fs1/oracle/prod
$ export PATH=$ORACLE_HOME/bin:$PATH
$ export LIBPATH=$ORACLE_HOME/lib:$LIBPATH
$ sqlplus
```

```
SQL*Plus: Release 10.2.0.5.0 - Production...
```

Tool and Utility Parameter Conventions

Oracle tools and utilities generally take command-style parameters (or arguments) that specify processing options, inputs, outputs, and so on. Depending on the tool or utility, parameters may be positional, identified by keyword, identified by UNIX-style option letters, or some combination of these. How parameters are specified depends on where and how the tool or utility is invoked: different conventions apply to batch (JCL), TSO, and a z/OS UNIX shell. This section describes those conventions generally, without going into the particulars of individual tools or utilities. For details on specific tools and utilities, refer to [Chapter 4, "Oracle Tools and Utilities on z/OS."](#)

As described in the following sections, parameters are specified in the JCL PARM field, the TSO command line, the "parm" string of a TSO CALL command, or the command line of a shell. Data that are not parameters may also appear in these places, including C standard file redirection operators, discussed in the section ["Standard Files and Redirection"](#) on page 2-19, and a special notation indicating that parameter values be obtained from a file, discussed in the section ["Parameters in Batch \(JCL\)"](#) on page 2-6. When these things appear in the PARM, command line, and so on, they are processed and removed before the Oracle tool or utility program code receives control and are not "seen" by the tool or utility.

Parameters in Batch (JCL)

When you run a tool or utility using JCL, parameters are supplied using the PARM keyword of the JCL EXEC statement. All Oracle tools and utilities are built with the NOEXECOPTS option, which means LE runtime options cannot be supplied in the PARM data, so everything in the PARM string except for LE redirection operators is seen by Oracle software. There are several different ways to code PARM data in JCL; the most common, used in our examples, is to enclose the data in single quotes (apostrophes). When a parameter value must include single quotes, code two single quotes together to signify one single quote in the value. Refer to the *z/OS JCL Reference* for details and variations of PARM syntax.

Some programs (such as SQL*Loader) have a large vocabulary of parameters including ones that specify data set or HFS file names. z/OS limits the PARM field to 100 total characters which can easily be exceeded by parameters you want to supply. To accommodate this, Oracle provides the ability to place parameters into a data set or HFS file. This is indicated by coding "++" (two plus signs) followed immediately by a filespec, with no intervening blanks, inside the PARM string. Filespecs and related details are covered in the section ["File Types and Filespec Syntax"](#) on page 2-14.

The ++ operator can appear alone in the PARM string or it can appear before, after, or among normal parameters. Wherever it appears in relation to normal parameters is where Oracle logically inserts the parameter data from the specified file; this is

significant when ++ is used to supply positional parameters. More than one ++ operator can appear in a single PARM.

The filespec following ++ can designate an existing Physical Sequential (PS) data set, a member of a partitioned data set, an instream (DD *) data set, or an HFS file. If an HFS file is used the filespec must use unambiguous HFS syntax, discussed in ["File Types and Filespec Syntax"](#) on page 2-14, and the address space must be authorized for z/OS UNIX System Services. Tools and utilities in batch run with POSIX OFF, so ambiguous notation for the filespec is interpreted as a DD or data set name.

When ++ specifies a sequential data set or PDS member, the data set can have either fixed- or variable-length records and the logical record length (LRECL) can be anything up to 1028 (1024 data bytes) that will hold the longest single parameter. Parameters in the file must be separated by one or more blanks or by a record boundary (or a newline character in an HFS file). Parameter values are not quoted (any quotes are treated as part of the value) and they cannot contain blanks or span multiple records or lines. Sequence numbers or other non-blank non-parameter data must not appear in a parameter file. Also, LE redirection operators and the ++ operator itself cannot be specified in a parameter file.

Example 1

The following example is part of a jobstep showing a typical parameter file setup with SQL*Plus, using an instream data set. In this case, all parameter values are being supplied in the file, so only the ++ operator appears in the PARM.

```
//PLUS EXEC PGM=SQLPLUS, PARM='++//DD:PARMS'
//PARMS DD *
-L
scott/tiger@db1
@//DD:SQL
/*
```

When SQL*Plus executes, it sees three parameters: -L, scott/tiger@db1, and @//DD:SQL.

Example 2

The following example produces the same behavior as Example 1, but the parameters are arranged differently:

```
//PLUS EXEC PGM=SQLPLUS, PARM='-L ++//DD:PARMS'
//PARMS DD *
scott/tiger@db1 @//DD:SQL
/*
```

If an error occurs when opening or reading a parameter file, descriptive messages are displayed and tool or utility initialization fails.

Some Oracle utilities, such as Export and Import, have their own parameter file feature that is part of the generic Oracle Database product. This is distinct from the ++ operator, which works with all Oracle tools and utilities and is unique to z/OS.

Parameters in TSO

In native TSO (not a TSO session that has entered a z/OS UNIX shell) there are two ways to invoke an Oracle tool or utility: as a true TSO Command Processor (CP) or through the TSO CALL command. Either can be used from plain TSO (the READY prompt), from an enhanced command interface such as ISPF or Session Manager, or

from a CLIST or REXX procedure. For simplicity's sake, our examples here show the READY prompt.

When the CALL command is used, parameters are passed in a single-quoted string similar to the JCL PARM field and subject to the same 100-character limit. Coding considerations in this case (including use of the ++ operator) are the same as those discussed for batch jobs in the prior section.

Example 1

The following example shows TSO CALL being used to run Oracle Export with a ++ operator for all parameters except the user ID and password. The parameters are in member EXP1 of a PDS named *tsoprefix*.MYORA.CNTL. The ++ operator specifies the data set name directly, instead of allocating a DD.

```
READY
alloc file(ora$lib) da('oracle.v10g.mesg') shr
READY
call 'oracle.v10g.cmdload(exp)' 'scott/tiger ++//myora.cntl(exp1)'
Export: Release 10.2.0.5.0 - Production...
```

To invoke a tool or utility as a TSO CP you must have the required program objects (generally, the CMDLOAD data set) specified in the logon procedure STEPLIB, in a linklist or LPA library, or in a TSOLIB command. In the CP case, parameters are specified after the command program name, separated by one or more blanks, for as long as necessary. If your parameters exceed the available terminal input area, end the line with a minus sign (-) or plus sign (+) character to indicate continuation, press ENTER, and resume typing parameters in the next provided input area. Refer to z/OS TSO/E documentation for full details on command line continuation.

Although it is less necessary (because there is no real parameter length limitation), the ++ operator can be used in TSO CP invocation as well.

Example 2

The following example shows the use of TSOLIB to establish an Oracle CMDLOAD data set as a source for commands, followed by a call to Oracle Export (as a CP) with all parameters included on the command line. Although the normal name for Export is EXP, it has been installed with the name ORAEXP to avoid conflict with IBM's export command. TSO continuation is used, in this case in the middle of the FILE= parameter value.

```
READY
alloc file(ora$lib) da('oracle.v10g.mesg') shr
READY
tsolib act da('oracle.v10g.cmdload')
READY
oraexp scott/tiger@db1 grants=n file=//ORADB1.EXP1-
.DATA' tables=(emp,dept)
Export: Release 10.2.0.5.0 - Production...
```

In this case, the third parameter seen by Export is //ORADB1.EXP1.DATA'.

Parameters in a z/OS UNIX Shell

In a z/OS UNIX shell, Oracle tools and utilities are invoked as commands at the shell prompt or from within a shell script. While some details of syntax and processing differ across the various UNIX shells, considerations are largely the same as those described for other Oracle UNIX platforms. Special attention is required when parameters contain characters that are subject to interpretation by the shell, such as \$

(dollar), * (asterisk), and both single and double quotes. Refer to the documentation for the specific shell you are using to understand its particular conventions and requirements. For the IBM default shell, `/bin/sh`, refer to *z/OS UNIX System Services Command Reference*, or type `man sh` at the shell prompt to display the "man pages" for the shell.

The ++ operator described for batch and TSO is not available in z/OS UNIX shell environments.

In this example, Oracle Export is invoked at the shell prompt with the same parameters as the preceding TSO example. The shell escape character (backslash) is used on the single quotes in the file parameter to keep the shell from interpreting them.

```
$ exp scott/tiger@db1 grants=n file=/'ORADB1.EXP1.DATA\' tables=(emp,dept)
```

Parameters Containing Spaces

Normally, tool and utility parameter values do not contain spaces but are separated by one or more spaces. There are, however, situations where a parameter containing spaces must be supplied, notably when using certain Oracle utilities and supplying the special logon identifier required for database control operations, which is `/ AS SYSDBA`. To specify such a parameter using PARM or command line, enclose it in full (double) quotes. Depending on the environment, LE or the POSIX shell will remove the quotes and present the enclosed data as a single parameter to the tool or utility. In a batch SQL*Plus job, for example, this looks like the following:

```
//PLUS EXEC PGM=SQLPLUS,PARM=' "/ AS SYSDBA "'
```

In a TSO command line setting it looks like the following:

```
READY
sqlplus "/ as sysdba"
```

Parameters containing spaces cannot be supplied within a parameter file specified with "++".

Environment Variables

Environment variables are simply named runtime entities with assigned character string values. Though they originated in the UNIX world they are also associated with C programming and were implemented in MVS C runtime services before MVS OpenEdition existed. Various Oracle components and z/OS components (particularly LE) use environment variables as parameters or to control internal processing options. You may need to specify certain environment variables when running Oracle tools or utilities or your own Oracle-accessing applications.

This section describes the mechanisms for specifying environment variables for Oracle software running in z/OS environments other than CICS TS and IMS TM. Refer to the *Oracle Database System Administration Guide for IBM z/OS on System z* for information about environment variables in CICS TS and IMS TM transactions.

Information about specific environment variables, their names, usage, and allowed values is provided in other sections of this manual and in the generic Oracle product documentation. In addition, environment variables specific to Oracle Database for z/OS are listed in Appendix A.

Environment Variables in a z/OS UNIX Shell

The environment variable concept is indigenous to UNIX and to the shell environment. The default shell on z/OS provides various mechanisms for setting environment variables either explicitly or automatically on shell startup. When you run an Oracle tool, utility, or other application in z/OS UNIX shell you must ensure that required environment variables are set. Typically, the program or script runs in a new shell process that is a subshell of the one launching the application. In this case, environment variables that were set in the launching shell must be exported to be passed to the application shell. In the default IBM shell, the `export` command does this. Our example shows the setting and exporting of two environment variables before invoking SQL*Plus at the shell prompt:

```
$ ORACLE_SID=PDB1
$ export ORACLE_SID
$ export LIBPATH=$ORACLE_HOME/lib:$LIBPATH
$ export PATH=$ORACLE_HOME/bin:$PATH
$ sqlplus scott/tiger
```

```
SQL*Plus: Release 10.2.0.5.0 - Production...
```

For complete details on using environment variables in the default IBM shell, refer to the *z/OS UNIX System Services Command Reference*, or type `man sh` at the shell prompt to display the "man pages" for the shell.

It also is possible for your own C/C++ application programs running in a POSIX environment to set or change environment variables using calls to runtime functions such as `putenv()`. This is discussed in the section "[Setting Environment Variables in a Program](#)" on page 2-12.

Environment Variables in Batch and TSO Environments

When an Oracle tool, utility, or user-written application runs as a batch job or TSO command with LE POSIX OFF, several different mechanisms are available for specifying environment variables. More than one mechanism can be used in a single application and it is legal for the same environment variable to be specified in more than one place. The last setting of a given variable is the one that takes precedence. The following lists the available mechanisms in the order in which they are processed:

1. The LE runtime option ENVARn (not available in Oracle tools and utilities)
2. A file of settings specified by the LE environment variable `_CEE_ENVFILE` normally set using LE runtime option ENVAR (not available in Oracle tools and utilities)
3. A global default set of Oracle environment variable assignments in a specific z/OS data set or HFS file
4. A local set of environment variable assignments specified by the `ORA$ENV DD` statement in the application
5. Program calls to functions such as C/C++ `putenv()` during application execution (discussed in "[Setting Environment Variables in a Program](#)" on page 2-12.

The first two mechanisms are provided by LE and are described in the IBM manual *z/OS C/C++ Programming Guide*.

The third and fourth mechanisms are provided by Oracle software and are available in Oracle tools and utilities and your own programs that access Oracle. The global environment file generally is set up by the system administrator, database administrator, or a person responsible for the Oracle product configuration on z/OS. It is meant to contain default environment variable assignments for all Oracle-accessing

applications in a sysplex. The global environment file is optional and may not be present in your system; if it is, it is used automatically without action on your part. Check with your Oracle or z/OS system administrator to learn if the global environment file is configured on your system.

The local Oracle environment variable file, specified by the ORA\$ENV DD statement, also is optional. Use it to supply environment variables specific to your job or application and to override (change) settings from the global file. The content and processing details for both the local and global files are described in the section "[Oracle Environment Variable Files](#)" on page 2-11.

Processing for the Oracle-specific mechanisms (prior steps 3 and 4) occurs during program initialization in the case of Oracle tools and utilities. In user-written Oracle applications it occurs when the Oracle program interface code initializes, typically on the first call to an Oracle interface function. This in turn depends on application design and can be early or late in the life of the application program.

As mentioned earlier, both Oracle software and z/OS software (particularly LE) make use of environment variables. If your application requires setting of environment variables for LE or other IBM components, use one of the mechanisms provided by LE (prior steps 1 and 2). Do not use the Oracle-provided mechanisms to set IBM-specific environment variables because the setting may occur too late, after IBM software has queried the variable. This is particularly the case with variables that affect LE enclave initialization. For an obvious example, using the local ORA\$ENV file to specify a value for `_CEE_ENVFILE` has no effect.

The converse—using LE mechanisms to supply Oracle-specific environment variables—is permitted. However, you might want to avoid LE's `_CEE_ENVFILE` mechanism for the reasons discussed at the end of the next section.

Oracle Environment Variable Files

This discussion pertains to both the global and local environment variable file. These can be a Physical Sequential (DSORG=PS) data set, a member of a partitioned (DSORG=PO) data set, or an HFS file in the POSIX file system. The local file, specified by an ORA\$ENV DD statement, can also be an instream (DD *) data set and can concatenate multiple data sets with compatible DCB attributes. When a sequential data set or PDS member is used, the data set can have either fixed- or variable-length records (RECFM of F, FB, FBS, V, or VB) and the logical record length (LRECL) can be anything up to 1028 (1024 data bytes). Choose an LRECL that will hold the longest required variable assignment statement you need.

In the environment file, each record (in a data set) or each line (in an HFS file) is either a comment or a variable assignment statement. Lines or records whose first nonblank character is # (pound), * (asterisk), or / (slash) are treated as a comment and are ignored. Records or lines that are entirely blank also are ignored. Otherwise, a record or line must contain a single assignment statement of the form

```
name = value
```

where *name* is the environment variable name (case-sensitive), "=" is the equals sign as shown, and *value* is as discussed in the following section. Blanks can precede and follow both the name and the value and are ignored. No other nonblank data can appear between the value and the end of the record or line. The entire statement must fit on one record or line: spanning or continuation is not permitted.

The value part is optional. If none is specified, the environment variable has an empty string as its value. Otherwise, the value is specified as an unquoted or quoted string. If it is not quoted, it starts at the first nonblank character after the "=" and ends at the

next blank or at the end of the record or line, whichever comes first. Either apostrophes or full quotes can surround a value to allow it to include blanks. In this case, the value begins with the character following the opening apostrophe or quote and ends at the matching (closing) apostrophe or quote, which is required and must be the same kind as the opening character. If a quote of the same kind is to be included as part of the value, place two quotes together. An imbedded quote of the other kind is not doubled.

Unlike in a shell, assignment statements in an environment variable file cannot dereference another environment variable. For example, given file statements like

```
That = something
This = stuff:$That
```

the value of environment variable `This` is the string `stuff:$That`, not `stuff:something`.

The following illustrates a local (`ORA$ENV`) file specified as an instream data set in a batch job. The variables and values are not realistic but were chosen to illustrate syntax variations.

```
//ORA$ENV DD *
***
*   Sample Oracle environment variable file
***

This_File='/'/'ORA.SOME.DATA.SET''
  VAR1   =      some_thing_1
VAR2 = "Don't go there."
VAR3 =
/*
```

As mentioned earlier, both the global and local environment variable files are optional. If the attempt to open either file fails with an error such as "File not found" or "DD statement not found", the error condition is ignored and processing proceeds normally. Other errors—including security rejects on file open, I/O errors reading the file, and syntax errors in the file contents—are reported through error messages. In this situation Oracle tool or utility initialization fails. In user-written applications, a user ABEND is issued by the Oracle interface stub.

There are significant differences between Oracle's environment variable file features and LE's `_CEE_ENVFILE`. The syntax requirements for the LE file are both simpler and more restrictive (and make fixed-length record formats all but unusable). By default, no error messages are displayed for errors that arise during LE environment file processing. This can make it difficult to detect problems related to environment variables. For non-POSIX applications, Oracle recommends using the Oracle-provided mechanisms to specify Oracle-specific environment variables.

Setting Environment Variables in a Program

If you create an Oracle-accessing applications using IBM C/C++ (Pro*C/C++, OCI, or OCCI applications) you have access to functions like `putenv()` and `setenv()` and to the `environ` pointer to set or change the values of environment variables.

In general, Oracle discourages programmatic manipulation of Oracle-specific environment variables. The exact "when" (and how often) Oracle software retrieves a given variable's value is an internal behavior and is not necessarily constant over even minor maintenance changes. An application that succeeds with this technique today might not work in a different version, release, or maintenance level of Oracle software.

Files, File Name Syntax, and File Processing

All Oracle tools and utilities, the Oracle database server, and in some cases the Oracle program interface code require access to data on disk or, sometimes, tape. z/OS presents challenges in this area due to the variety of distinct file types and the associated software interfaces (Access Methods or APIs). To meet the needs of z/OS users across the various execution environments, Oracle tools, utilities, and program interface software must be able to process POSIX HFS files and also sequential, partitioned, JES spool, and in some cases VSAM data sets. As a user of these components you must understand how files are specified (when you supply file information to a component) and how the software processes that information.

IBM C/C++ and LE provide most of the underlying file processing for end-user-accessible Oracle features. It is helpful to understand how LE processes the various types of files and data sets in z/OS. The *z/OS C/C++ Programming Guide* and *z/OS C/C++ Run-Time Library Reference* are the definitive sources of information about LE file processing. Some details on LE processing or behavior are provided here as a convenience, but the IBM manuals are the definitive source of information.

Filespecs

Central to an understanding of file processing is the notion of a filespec, which is our term for a character string that wholly or partially identifies a file. You will supply filespecs to Oracle components in parameters, commands, and possibly in SQL, PL/SQL, or server Java statements. On z/OS, a filespec can specify a data set or DD name, an HFS path and file name, a SYSOUT (spool) data set, or the terminal (in TSO or a z/OS UNIX shell). Which of these a given filespec means depends on the syntax of the filespec and on the POSIX indicator. The latter consideration is significant: as you will see shortly, a filespec like `myexport.dat` can mean a data set in one execution environment and an HFS file in another.

Oracle generally uses the syntax defined by IBM C/C++ and LE for filespecs. In some areas, notably SYSOUT support, Oracle augments LE filespec syntax to provide useful capabilities not provided by LE. Optionally, Oracle also supports a subset of the filespec syntax used by the Oracle C runtime library so that existing customer JCL, scripts, and procedures that were developed for Oracle C runtime continue to work without changes. The Oracle C runtime syntax overlaps valid LE syntax and so is not recommended for new customers and new applications. Details on this compatibility feature are in the section "[Oracle C Runtime Compatibility](#)" on page 2-29.

As suggested earlier, the filespecs you supply to Oracle functions are not always complete. This applies to filespecs that designate a data set name or an HFS file name. In some components, these names are subject to extension, which means adding a suffix string to the name. Usually the suffix indicates the type or usage of the data in the file, such as `.sql` or `.SQL` for a file containing SQL statement text. The extension mechanics for HFS file names and data set names differ; in the case of data sets, the extension mechanism also can be used to associate attributes (space and DCB attributes) with a data set that is being created.

Other types of file name augmentation or manipulation also occur. LE adds the PROFILE PREFIX (in TSO) or the z/OS user ID (in other environments) to the beginning of any filespec data set name that is not enclosed in apostrophes. Certain Oracle components construct filespecs by combining separate "directory" and "name" pieces. Oracle utilities, particularly SQL*Loader, may derive one filespec from another by removing one suffix extension and adding a different extension. A z/OS-specific Oracle feature called FNA can be used to map simple file names (as seen by Oracle) to

member names in a z/OS partitioned data set. These topics are covered in the sections that follow.

File Types and Filespec Syntax

Before we describe the syntax and interpretation of different types of filespecs we have to establish some more terminology. Most filespecs and types are usable in all of the client environments: batch, TSO, and z/OS UNIX shell. CICS TS and IMS TM are discussed separately in [Chapter 4, "Oracle Tools and Utilities on z/OS."](#) A few types are not usable in obvious circumstances: you cannot specify a "TSO terminal" filespec for an input file in a non-TSO address space. More important, some filespecs are interpreted differently depending on the POSIX indicator; with POSIX ON they refer to data sets and with POSIX OFF they refer to HFS files. We call these ambiguous filespecs.

Ambiguous filespecs are perfectly legal and acceptable to use except where the POSIX-determined behavior is not what you want. If you want to refer to a data set in a POSIX ON application or to an HFS file in a POSIX OFF application, you must use an unambiguous filespec, one which is self-identifying as to type. C/C++ and LE use specific syntax elements to make a filespec unambiguous. If a filespec begins with exactly two slashes (the third character is not a slash), it is taken as a non-HFS filespec regardless of POSIX. If it does not begin with exactly two slashes but it contains one or more slashes (anywhere in the filespec), it is taken as an HFS filespec regardless of POSIX. If neither case applies, the filespec is ambiguous. There are some subtle details in this scheme when DD name filespecs are used. Refer to the section "[DD Names](#)" on page 2-15.

The treatment of ambiguous filespecs sometimes counters intuition. The criteria described previously mean 'JSMITH.TEST.SQL(CASE1)' is an ambiguous filespec. The fact that it looks very much like a z/OS partitioned data set and member name does not matter. If this filespec is used for output in a POSIX ON setting, an HFS file with a peculiar 24-character name-including the apostrophes-is created. One of the interesting attributes of the POSIX HFS is that virtually anything but a slash is legal within a file name, including blanks, punctuation, and even non-printing characters. When POSIX is ON, an ambiguous filespec is treated as HFS no matter how data-set-like it appears.

The converse treatment is equally impartial. You might use the filespec =Silly_QA_Tests=.sql in a command in a batch (POSIX OFF) SQL*Plus job expecting the software to recognize that the filespec is valid only as an HFS file name, and treat it as such. Instead, you will get an error reporting an invalid (data set) file name.

Data Set Names

The full form for specifying a z/OS data set by its name is

```
// 'dsname'
```

or

```
// 'dsname(member)'
```

where *dsname* is the 1-character to 44-character fully qualified data set name and *member* is a 1-character to 8-character member name for a partitioned (DSORG=PO) data set or a numeric Generation Data Group (GDG) index (possibly beginning with + or -).

The pair of slashes signify that what follows is not an HFS file name. They are required to access a data set in a POSIX ON environment. When POSIX is OFF they can be omitted.

The apostrophes (single quotes) indicate that *dsname* is a complete data set name not subject to LE prefixing. If you omit the apostrophes, LE prefixes *dsname* with the PROFILE PREFIX (in TSO) or with the z/OS user ID (in all other environments). (If you have enabled Oracle C runtime compatibility, prefixing occurs only in TSO.) If a TSO session has PROFILE NOPREFIX set, no prefix is added. Apostrophes also affect Oracle's tendency to add a suffix extension to a data set name; this is discussed later in the section "[File Name Manipulation](#)" on page 2-21.

The *dsname* and *member* portions of the filespec are case-insensitive: lowercase letters are converted to upper case. To be usable, a data set name must ultimately have two or more name segments or qualifiers. The name in the filespec can be a single qualifier if prefixing and/or suffix extension applies to supply a second qualifier. If apostrophes are used to prevent prefixing (or if PROFILE NOPREFIX is set in TSO), the name in the filespec normally must have two or more qualifiers.

A data set referenced in this fashion can be physical sequential (DSORG=PS), partitioned (DSORG=PO), or VSAM (ESDS, KSDS, or RRDS). Use of VSAM is limited to certain Oracle components or features discussed in later chapters. If the filespec is being used for output purposes and the data set does not exist, it is created through z/OS dynamic allocation as a non-VSAM (PS or PO) data set. (Creation of VSAM data sets is not supported.) Dynamic creation relies on installation default mechanisms to determine SMS classes (or unit and volume) and disk space allocation. These may or may not be appropriate for your application and you may want to consider preallocating such data sets and/or using a DD filespec, discussed later. Space allocation for some types of files can also be influenced using FNA, described later.

For an input file it is legal to specify a partitioned data set name with no member name. Doing so, however, accesses the directory data of the PDS rather than any member. There currently are no Oracle components that exploit this capability.

The following are examples of valid data set name filespecs.

```
// 'JSMITH.ORAEXP.TEST.DAT'  
prod.sql(case1)  
'ODB1.init.ora'  
//discard1  
//SAMPLE.LOG
```

In the prior set of examples, the first, fourth, and fifth are unambiguous.

DD Names

You can identify a file as a DD statement or TSO allocation using the following filespec forms:

```
//DD:name
```

or

```
//DD:name(member)
```

where *name* is the 1-character to 8-character DD name and *member* is a 1- to 8-character member name when the DD statement or allocation is a partitioned (DSORG=PO) data set.

The pair of slashes signify that what follows is not an HFS file name. When POSIX is OFF they can be omitted. If they are omitted when POSIX is ON, LE does something

surprising: it checks to see if the specified DD name is allocated in the job or address space. If it is, the DD is opened. If it is not, an HFS file named `DD:name` or `DD:name(member)` is opened. This is one case where the interpretation of an ambiguous filespec depends on more than just POSIX.

The DD: portion of the filespec is required to indicate that what follows is a DD name. It is case-insensitive and can use any of the 4 possible combinations of upper case and lowercase "D." The name and member parts are also case-insensitive and are converted to upper case.

The DD statement or TSO allocation specified by this filespec can be any of the following:

- Physical sequential (DSORG=PS) or partitioned (DSORG=PO) data set
- VSAM KSDS, ESDS, or RRDS cluster
- Instream data for input usage (DD *) or SYSOUT for output usage
- Dummy (DD DUMMY)
- TSO terminal (DD TERM=TS or TSO ALLOCATE DA(*))
- HFS file (DD PATH=)

If the file is being used as input, the DD can be a concatenation of multiple data sets of compatible type and DCB attributes.

When a partitioned data set is used, either the filespec or the DD statement (but not both) can include the parenthesized member name. If neither specifies a member name, the PDS directory data is accessed rather than any member; this is allowed only with input usage.

Files specified as DD name filespecs have some specific behaviors in Oracle software. Name manipulation operations such as extension have no effect on a DD filespec nor can such a filespec be parsed or divided into logical pieces. This affects the way some Oracle components work when using DD filespecs. This is covered in more detail in ["File Name Manipulation"](#) on page 2-21 and in the later chapters dealing with specific Oracle components.

The following are examples of DD name filespecs:

```
//dd:mysql
DD:LIB1(sample)
dd:x
//DD:REPORT
```

In the prior set of examples, the second and third are ambiguous; when POSIX is ON, they are subject to the processing discussed earlier.

SYSOUT

Oracle provides a logical extension of LE file syntax for direct specification of SYSOUT data sets for output. This is usable in most places where Oracle requires an output file specification but it is associated primarily with text (message or report) files. The filespec syntax is

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

where *class* is the 1-character SYSOUT class, *form* is a 1-character to 4-character form name, and *dest* is a 1-character to 8-character JES destination identifier.

The pair of slashes signify that what follows is not an HFS file name. They are required to use a SYSOUT filespec in a POSIX ON environment. When POSIX is OFF they can be omitted.

The *SYSOUT*: portion of the filespec is required to indicate that what follows is a SYSOUT specification. It is case-insensitive and can use any combination of upper- and lower-case letters. It also can be abbreviated to *S*: or *s*:

The *class* is a valid JES SYSOUT class (letter or digit), or an * (asterisk) signifying that the job's MSGCLASS be used. If the class is omitted it defaults to *. If you omit the class but want to supply a form or destination, code a single comma.

The *form* is a valid JES form name. The default, when form is omitted, is installation-dependent. If you omit the form name but want to supply a destination, code a single comma.

The *dest* is a valid JES destination identifier. Legal destination values and the default depend on whether your z/OS system uses JES2 or JES3.

Check with your z/OS system administrator for specific information about output classes, form names, and destinations used in your installation.

Like DDs, SYSOUT filespecs are immune to file name manipulation operations such as extension and parsing. Here are some examples of SYSOUT filespecs:

```
//SYSOUT:G,,JSMITH
s:
sysout:*,STD
//S:,,HQPRT1
```

The second and third examples are ambiguous and would result in oddly-named HFS files in a POSIX ON application.

TSO Terminal

In a TSO session you can specify both input and output files as the terminal device using the filespec

```
//*
```

The slashes signify that what follows is not an HFS file name and the single asterisk is required to indicate the TSO terminal. The slashes can be omitted when POSIX is OFF.

Optionally, you can follow the asterisk in this filespec with either a data set name (with or without surrounding apostrophes) or a DD name specification beginning with DD:. When such a filespec is opened in a non-TSO address space, it is processed as though the asterisk was not there-the indicated data set or DD name is opened. This enables you to code a single filespec that uses the TSO terminal under TSO and a data set in other environments. You cannot specify an HFS file name after the asterisk, but a DD name specification can refer to an HFS file (DD PATH=).

If the TSO terminal is specified for an output file in a non-TSO address space and no data set or DD name follows the asterisk, LE allocates a SYSOUT data set with default class, form name, and destination. For an input file, this situation results in an error.

Unless it includes a data set name as just described, TSO terminal filespecs are immune to name extension and other name manipulation operations. Manipulating the data set name included on a TSO terminal filespec may or may not be meaningful depending on the usage and environment.

When an input file is specified as the TSO terminal, the keyboard is unlocked to accept user input. Whatever is typed up to the point of pressing the ENTER key is presented

to the software as a single logical record. To signal the end of input (EOF), type /* (slash and asterisk) alone as input and press ENTER.

TSO terminal support in LE uses the TGET and TPUT services to read and write at the terminal. Using TGET for input means that the terminal is always accessed; so-called stacked input (the TSO input stack) is not processed by TGET. Applications cannot place data on the input stack (for example, using REXX "queue") and have it read by an Oracle component through a TSO terminal filespec. Using TPUT for output means that the terminal is always the destination. This means features like REXX OUTTRAP cannot be used.

The following are examples of TSO terminal filespecs:

```
/*
*dd:print
*
/* 'JSMITH.ORA1.CNTL(SQL1)'
```

HFS Files

The syntax for HFS filespecs is that which is typical of UNIX systems. A complete file specification, or path, begins with a slash (signifying the root directory). This is followed by zero or more directory names, each ended with a slash, and finally the file name. The sequence of directory names represents navigation through the logical hierarchy represented by the directories. A single period is interpreted as the current directory in the navigation and a pair of periods refers to the parent (previous) directory. Besides directories and files, the HFS supports links, which are essentially aliases or alternative names for directories and files.

When an HFS filespec begins with a slash it is said to be absolute, meaning the entire navigation through the HFS is specified. If it does not begin with a slash it is taken as being relative to the current working directory, which is an attribute of the running process. Typically, when you are in a z/OS UNIX shell, the current working directory starts out set to your user ID's home directory. You can change the current working directory with the `cd` shell command and other mechanisms.

File names in the HFS are essentially unconstrained in content; while there are conventions for naming files there are few system-imposed rules. Names are case-sensitive, can contain almost any characters (including non-printing characters), and can be quite long: the total length of all slashes, directory names, and finally the file name in a z/OS HFS filespec can be up to 1023 characters. Here are some examples of valid HFS filespecs:

```
/u/jsmith/test.sql
../oradev/c/rover/rover1.c
@-@-@ Read_Me_And_Weep @-@-@
SYS1.PROCLIB(ASMHCL)
```

From a UNIX perspective, the first two are fairly conventional while the latter two are somewhat unusual but still legal. As the last example shows, valid HFS name syntax is a superset of valid z/OS data set name syntax.

For complete information about z/OS HFS files, refer to the *z/OS UNIX System Services User's Guide* and related IBM publications.

In a POSIX ON application, any filespec that does not begin with exactly two slashes is interpreted as an HFS filespec. There is one exception in the case of DD filespecs, described in the section "[DD Names](#)" on page 2-15.

When POSIX is OFF, a filespec must contain at least one slash (and not begin with exactly two slashes) to be recognized as HFS. In this situation, if you want to refer to a

file relative to the current working directory (where no directory names and slashes are needed), you must include some unnecessary but benign syntax:

```
./test.sql
```

This is a reference to the file `test.sql` in the current directory, signified by `./`. The `./` part is not really needed to get to the file but it gets a slash into the filespec.

Besides regular files, HFS structure and syntax is used for various special files in z/OS UNIX System Services. Many of these appear under HFS directory `/dev`, which provides an HFS structure for special devices used as files. Two of these that are often used by shell-based applications are `/dev/tty`, which indicates the terminal associated with the current process, and `/dev/null`, which indicates a null or empty file (similar to z/OS DD DUMMY).

Standard Files and Redirection

C language applications, including Oracle tools and utilities, are provided with three pre-opened files by the runtime environment. (These are called standard streams in IBM terminology.) One is an input file (standard input) and the other two are outputs (standard output, usually used for normal output, and standard error, usually reserved for error messages and supporting data). Most Oracle tools and utilities get their primary input (such as control statements) by reading from standard input and display their normal output (such as status messages and listings) by writing to standard output.

What is actually opened as standard files depends on the environment. In a z/OS UNIX shell, the standard files are controlled by the shell; they normally are assigned to the terminal associated with the process but they might be changed using shell mechanisms such as redirection (which specifies a different filespec for a standard file) or piping (where the standard output of one command is treated as the standard input to another). For details on standard files in shell sessions refer to the *z/OS UNIX System Services User's Guide* and to the documentation (such as "man pages") for the particular shell you are using.

When not in a shell, the standard files normally default to the terminal (in TSO) or to certain DD names (if present) or to SYSOUT (if not, in the case of standard output and standard error). Oracle tools and utilities deviate from normal LE processing in this area. This is done mainly to be compatible with past (Oracle C runtime) behavior, but it cannot be disabled with an environment variable like other Oracle C runtime compatibility features because the processing occurs before environment variables can be queried.

Redirection, meaning respecification of standard files to something besides the default, is also available in TSO and batch using command line or PARM syntax similar to that in shells. When redirection operators are included in the command line or PARM field they are processed during LE initialization and are not "seen" by the Oracle tool or utility. Redirection conventions and syntax are described in ["Redirection Operators in TSO and Batch"](#) on page 2-20.

Standard File Defaults (non-POSIX)

When you do not redirect standard files for an Oracle tool or utility running in TSO or batch (which is usually the case), the default behavior differs slightly between the two environments. In TSO, the default behavior is as follows:

- Standard input defaults to the TSO terminal unless a SYSIN allocation (DD) is present, in which case SYSIN is read.

- If both SYSOUT and SYSERR allocations (DDs) are present, standard output and standard error are written to SYSOUT and SYSERR respectively; otherwise, both are written to the TSO terminal.

In a batch job, when the standard files are not redirected, the following defaults are used only if all 3 of the DD names SYSIN, SYSOUT, and SYSERR, are present:

- Standard input defaults to the SYSIN DD.
- Standard output defaults to the SYSOUT DD.
- Standard error defaults to the SYSERR DD.

This behavior mimics Oracle C runtime, which required all three of the standard file DDs to be provided to initialize successfully. If any of these three DD names is not included in the JCL, the job uses the normal IBM C/C++ defaults instead, as follows:

- Standard input reads the SYSIN DD if provided, otherwise standard input is treated as empty (like DD DUMMY).
- Standard output goes to DD SYSPRINT if provided, or to SYSTERM if provided and SYSPRINT is not, or to SYSERR if provided and neither SYSPRINT nor SYSTERM is provided. If none of these DDs is provided, standard output is dynamically allocated to the default spool output class and form.
- Standard error goes to the LE message file (normally the SYSOUT DD) if it is provided, otherwise it is dynamically allocated to the default spool output class and form.

Be aware that there is a logical conflict between Oracle C runtime compatibility and IBM C/C++ normal processing in certain situations. For example, if a batch job supplies SYSIN, SYSERR, and SYSOUT DDs intending them to be used as standard input, output, and error respectively (per the C/C++ defaults just described), the job will assume Oracle C runtime compatibility is intended and will reverse the roles of SYSERR and SYSOUT to be standard error and standard output.

Typically, when setting up JCL to run an Oracle tool or utility in batch, you will supply a SYSIN DD statement for whatever data the particular tool or utility expects as its primary input. The exact form or syntax of that data depends on the particular tool or utility. Some components, such as SQL*Loader, do not necessarily use standard input, in which case the SYSIN DD can be omitted. For the output files, if the fallback SYSOUT dynamic allocation is acceptable, you need not supply the associated DDs. You will need to supply them if you want to have a different SYSOUT class or other characteristics, or if you want to direct the output to something other than SYSOUT, such as a disk data set.

Redirection Operators in TSO and Batch

You can modify the defaults for standard files using an LE redirection operator or one of the Oracle C runtime redirection operators for which backward compatibility is provided. Redirection is coded on the command line or in the PARM field but it is not processed by the Oracle tool or utility and does not affect tool or utility parameters. You cannot place a redirection operator in a parameter file specified with "++".

Redirection syntax consists of an operator (which can be one or several characters long) followed by a filespec with no intervening spaces. There also are special redirection operators to indicate that standard output and standard error are written to the same file. Complete details on redirection operators are in IBM's z/OS C/C++ Programming Guide. The operators most commonly used are as follows:

- "<" for standard input redirection

- ">" for standard output redirection
- "2>" for standard error redirection;
- "?" is also allowed for Oracle C runtime backward compatibility

The filespec that follows a redirection operator can be a data set name, DD name, TSO terminal, or HFS type. Depending on POSIX, you may need to use unambiguous filespec syntax to ensure that the correct type of file is accessed. Redirection cannot use the SYSOUT filespec type described earlier because SYSOUT filespecs are an Oracle extension that LE does not support directly.

When you use redirection, the filespec you supply must be capable of being opened in the correct mode (input or output) at tool or utility initialization time; there is no fallback to alternate DD names or to dynamic allocation like there is for default standard files. If the filespec for a redirection cannot be opened, LE initialization issues error messages followed by a user ABEND.

The filespec following a redirection operator is permitted to use certain Oracle C runtime syntax for backward compatibility. Specifically, the prefixes /DD/, /ODD/, /DSN/, and /ODSN/ are accepted and converted internally to LE equivalents before the file is opened. This backward compatibility cannot be disabled. Be aware that this makes it impossible to use redirection with an HFS file whose directory path begins with /DD/, /ODD/, /DSN/, or /ODSN/ in any mix of letter case.

The following example shows a portion of a SQL*Plus jobstep that includes redirection of standard output. It also supplies the parameter /nolog to SQL*Plus followed by other parameters that are in a parameter file which is a member of a PDS.

```
//PLUS EXEC PGM=SQLPLUS,
// PARM='>' 'JSMITH.PLUSTEST.OUT' ' /nolog ++dd:moreparm'
//STEPLIB DD DISP=SHR,DSN=ORACLE.V10G.CMDLOAD
//ORA$LIB DD DISP=SHR,DSN=ORACLE.V10G.MESG
//MOREPARM DD DISP=SHR,DSN=JSMITH.ORA.PARMS(PLUS1)
//SYSIN DD ...
```

Because this is a batch job execution of SQL*Plus, POSIX will be OFF and the ambiguous filespec 'JSMITH.PLUSTEST.OUT' will be treated as a data set name. JCL PARM syntax conventions require that the surrounding apostrophes be doubled so that LE sees single apostrophes in the filespec.

File Name Manipulation

Filespecs that you supply to Oracle may be subject to various kinds of manipulation or augmentation before they are actually opened for I/O operations. You must understand this behavior to know what to code for a filespec in a given application. Some behaviors are common across all Oracle components and others are specific to particular components. This section describes in general the different ways filespecs are processed; refer to later chapters for details on specific components, tools, or utilities.

What's in a Name?

Most kinds of manipulation or augmentation pertain to filespecs that designate either a data set name or an HFS file name. DD, SYSOUT, and TSO terminal filespecs without an included data set name are mostly immune to the processing described here.

Oracle views both data set names and HFS file names as being composed of several distinct logical parts: a directory (sometimes called a path or location), a base name,

and an extension. Some components accept separate directory and base name/extension strings and combine them before use. Others may parse or decompose a filespec you supply into these pieces and then use the pieces to construct new filespecs. In the simplest case, components often add or replace the extension part. How these operations work differs between data sets and HFS files.

Data Set Name Parts

For z/OS data set name filespecs, the logical parts are determined as follows:

- The leftmost (high-level) qualifier of a data set name is taken as the directory. If a filespec data set name is not surrounded with apostrophes (it is subject to prefixing with the TSO PROFILE PREFIX or the z/OS user ID), the directory is an empty string signifying that a directory is supplied by LE. If the filespec includes "/" or surrounding apostrophes, the "/" and "/" or a single apostrophe are included with the directory.
- If the data set name contains three or more qualifiers (two or more, when prefixing applies), the rightmost qualifier is taken as the extension. Otherwise, the extension is an empty string signifying "no extension."
- Everything that remains after the directory and extension are subtracted is considered the base name. When a data set filespec includes a member name or GDG index in parentheses, the parenthesized part is retained in the base name. If the original filespec included a "/" prefix, the base name also has a "/" prefix.

The example shows how various data set name filespecs are decomposed into logical parts. From left to right, the four parts are: filespec, directory, base name, and extension. Assuming POSIX is OFF, ambiguous filespecs are treated as data sets.

// 'JSMITH.TEST.SQL'	// 'JSMITH	//TEST	SQL
loader.ct1(demo3)	(empty)	loader(demo3)	ct1
// '*ORADB1.PAY.LST'	// 'ORADB1	// *PAY	LST
mydata	(empty)	mydata	(empty)
arproj.case.live.data	(empty)	arproj.case.live	data

When Oracle composes a data set name filespec from separate parts, slightly different content rules apply. A directory part that you supply can contain more than one data set name qualifier, it can include an ending period, and it need not have the "/" prefix. If what Oracle uses as the base name part includes apostrophes, neither the directory nor the extension part are added—such names are immune to alteration. The following list shows examples of various name pieces and how they are combined by Oracle. From left to right, the four parts are: directory, base name, extension, and result filespec.

// 'JSMITH	//TEST	SQL	// 'JSMITH.TEST.SQL'
'ARD07N.	lib.oracle(bb3)	CTL	'ARD07N.lib.oracle.ct1(bb3)'
(empty)	'PROD.ORA3.OUT'	lst	'PROD.ORA3.OUT'

HFS File Name Parts

When a filespec refers to an HFS file, Oracle components on z/OS follow the same rules as Oracle on other UNIX platforms for decomposing and composing filespecs. When decomposing an HFS filespec:

- Everything up to the rightmost slash is taken as the directory. The directory can be absolute (begins with a slash) or relative (begins with a subdirectory name, a single period, or two periods). If the filespec contains no slashes, the directory is an empty string which means the current working directory.

- Everything to the right of the rightmost period in the remaining text is taken as the extension. If there are no periods or the text ends with a period, the extension is an empty string.
- Whatever remains after the directory and extension are subtracted is the base name. It is possible (though unusual) for the base name to be an empty string.

Composing a filespec from these parts essentially concatenates them in order: directory, base name, and extension. An intervening slash is added between the directory and the base name if needed, and a period is added between the base name and the extension if needed.

Extension Processing

Most Oracle tools and utilities perform extension on filespecs before they are used. Extension associates a logical data type identifier with the filespec, such as "sql," for a file containing text SQL and PL/SQL statements. The extension identifiers come from Oracle software and usually (though not always) are three characters or less. In the case of filespecs that specify a data set or HFS file name, the extension operation may modify the name by inserting or appending the extension identifier text. The specific extensions used and processing done by each tool or utility is covered in later chapters of this manual and in the generic documentation. Here we cover the general behavior of the extension mechanism.

On z/OS, extension can also be used to completely transform the syntax of a filespec, simplifying the use of partitioned data sets with some components, and to associate disk space and DCB attributes with data sets being used for output. These z/OS-specific capabilities are described in the section "FNA" on page 2-24.

Extension applied to filespecs other than data set or HFS file names, such as DD, SYSOUT, and TSO terminal filespecs without an included data set name, does nothing to change the filespec.

With data set name filespecs, an extension is incorporated into the data set name if the name is not enclosed in apostrophes. The rationale for this is the same as that of LE, which avoids adding a user ID or PROFILE PREFIX to data set names enclosed in apostrophes. The extension mechanism adds the extension suffix to the right-hand end of the data set name (with an intervening period) regardless of the number of data set name segments already present. If a parenthesized member name or GDG index are present, they are preserved in the modified name. The following are examples of extension processing on various data set filespecs. From left to right, the three parts are: filespec, extension, and result.

//sample3	sql	//sample3.sql
'JSMITH.LOADER.CTL	log	'JSMITH.LOADER.CTL'
payroll(rcn41c)	lst	payroll.lst(rcn41c)
temp.acct.sql	sql	temp.acct.sql.sql

With HFS filespecs, the extension mechanism replaces a file name extension if one is already present; otherwise, the extension is added to the name with an intervening period. This differs from data set name processing and reflects the convention that most HFS file names do not have more than two segments separated by periods. The following are examples of the HFS filespec extension. From left to right, the three parts are: filespec, extension, and result:

sample3	sql	sample3.sql
./proj9/admin/load1	ctl	./proj9/admin/load1.ctl
/u/jsmith/test.c	lst	/u/jsmith/test.lst

FNA

On z/OS, the extension mechanism can be used to do more complex processing for data sets than just the addition of a name suffix. The facility that does this is called FNA (for File Name Augmentation). Using FNA enables you to do two things to a filespec associated with a given extension:

- Transform the filespec into different syntax before it is used. This is used primarily to cause certain Oracle components to treat simple 1-segment file names as PDS member names.
- Associate disk space and/or DCB attributes with a filespec that will be opened for output processing.

These two actions are independent of each other; when you use FNA, you can cause either or both of them to be applied to a given type of file; in other words, to a given extension.

To use FNA, you must supply an FNA control file using an ORA\$FNA DD statement in your job or TSO session. The DD can specify a sequential data set, a member of a PDS, or an HFS file. When you use a sequential or partitioned data set (which is usual) either fixed-length or variable-length record formats of any legal LRECL are permitted. If no ORA\$FNA DD is supplied, extension processing for data sets is simply that described in the previous section.

In the ORA\$FNA data set, member, or HFS file you supply one or more keyword statements that configure FNA. Each statement supplies FNA information for a single extension. The general form of the statement follows, where *ext* is an extension identifier, *newname* is a name transformation specification, and *attrs* is attribute specification data for output file usage:

```
FSA( FTYPE(ext) FNAME(newname) FATTR(attrs) )
```

The FSA, FTYPE, and at least one of FNAME or FATTR are required. Spaces are permitted before and after keywords, parentheses, and enclosed values. One FSA entry can span multiple records or lines if the break is not within a value. Comments are permitted, begin with "/" and end with "/" and can span multiple records or lines. Completely blank lines are ignored. Extraneous data (such as record sequence numbers) must not be included.

The FTYPE value *ext* is a 1-character to 8-character alphanumeric file extension identifier without a leading period. This indicates the specific extension to which the entry applies and should match an extension used by some Oracle component. Since FNA applies only to data sets (not HFS files) the extension is case-insensitive and can be given in upper- or lower-case letters. Only one FSA statement for a given FTYPE extension should appear in the ORA\$FNA file; duplicate entries for an extension produce a warning and are ignored.

FNA Name Transformation

The FNA keyword parameter FNAME value *newname* specifies a syntax transformation of the input filespec. It usually is specified as a character string enclosed in apostrophes or quotes. If you use apostrophes and want an apostrophe to appear in the value, code two apostrophes in succession. The transformed filespec is produced by copying this character string and replacing certain characters with new values:

- A "+" (plus sign) is replaced with all or part of the original input filespec text. If the + appears inside parentheses in the *newname* string and the input filespec contains a period, only the part up to the first period is used. Otherwise, the entire input filespec is substituted.

- An "*" (asterisk) is replaced with the TSO PROFILE PREFIX (in TSO) or with the z/OS user ID (in other than TSO). If PROFILE NOPREFIX is set in TSO, then * is replaced with an empty string.

After the string substitution is done, if the result is a data set name specification without surrounding apostrophes, the original extension suffix is added at the end of the data set name in the same fashion as normal extension processing described earlier. If the result is a DD or SYSOUT filespec, or a data set name surrounded with apostrophes, the original extension is discarded.

The primary use of the name transformation mechanism is to allow simple file names used by certain Oracle components to be treated as member names in a partitioned data set. This kind of transformation is done using FNA entries like

```
FSA( FTYPE(sql) FNAME('//DD:SQLLIB(+)' ) )
```

or

```
FSA( FTYPE(CTL) FNAME(''oradb1.prod.cntl(+)'') )
```

The transformation is truly mechanical as described previously, however, and can be used in other ways to impose name structure. The following examples show how the transformation applies to various combinations of input filespec and FNAME string. From left to right, the four parts are: input filespec, FNA FNAME, and result filespec.

catalog	//DD:SQLLIB(+)	//DD:SQLLIB(catalog)
temp.ora	'*.ORAPROD.LIB(+)'	'prefix.ORAPROD.LIB(temp)'
ldr1	ORACLE(+)	ORACLE.ct1(ldr1)
drc10a.c	'JSMITH.ORA.+'	'JSMITH.ORA.drc10a.c'

The third example in the prior list assumes that extension .ct1 is involved.

The filespecs produced by FNAME transformation can use Oracle C runtime syntax if Oracle C runtime compatibility is enabled. Refer to the section ["Oracle C Runtime Compatibility"](#) on page 2-29 for more information.

FNA Attribute Assignment

The FNA keyword parameter FATTR is used to associate an attribute string with files having a particular extension. It is pertinent only when the associated file is used for output and has no effect on input usage. The attributes that can be specified include disk space quantities and DCB characteristics (record format, logical record length, and block size). The disk space attribute pertains only to data set name filespecs for data sets that do not already exist (will be created when opened). DCB characteristics are meaningful for all output data set situations including data set name, DD;, SYSOUT;, and TSO terminal filespecs.

The FATTR attr string normally is enclosed in apostrophes or quotes. It must begin with a semicolon (;) followed by one or more keyword assignments separated by commas. All text following the semicolon ultimately is passed to the C/C++ runtime function `fopen()` in the "mode" argument. Although there are a number of options that can be included in the mode string, only the following keywords are supported by Oracle for use in FNA:

Table 2-1 FNA Keywords and Descriptions

Keyword	Description
recfm=	Specifies the record format. Record format choices are the familiar z/OS JCL RECFM terms such as V, VB, F, FB, and so on. You can also specify <code>recfm=*</code> or <code>recfm=+</code> to request preservation of attributes already assigned to the data set or coded on the DD statement. To understand how these work, refer to the section "Data Set DCB Attributes" on page 2-27.
lrecl=	Specifies the logical record length in bytes as a decimal integer. Choose a valid logical record length that meets any requirements specific to the tool or utility.
blksize=	Specifies the physical block size in bytes as a decimal integer. Choose a valid block size large enough to accommodate the logical record length. When using record format F or FS, block size must equal the logical record length. When using fixed blocked format (FB or FBS), block size must be an integer multiple of logical record length.
space=	Specifies space allocation for a new data set on disk. The syntax for the value is similar to that of the SPACE keyword in z/OS JCL: <i>(units, (pri, sec, dir))</i> where <i>units</i> is <i>cyl</i> for cylinders, <i>trk</i> for tracks, or a decimal integer block size for allocation in blocks, and <i>pri</i> , <i>sec</i> , and <i>dir</i> are decimal integer quantities for primary space, secondary space, and PDS directory blocks respectively. The units and a primary quantity are required. Directory space is required when creating a partitioned data set and prohibited otherwise.

For details on these keyword parameters and their allowed values and syntax refer to the *z/OS C/C++ Programming Guide*.

Data set DCB attributes can come from sources other than FATTR or they can be completely unspecified, resulting in default behavior. Refer to the section ["Data Set DCB Attributes"](#) on page 2-27 to understand how the various sources and default mechanisms determine what is ultimately established for a given data set.

Example FNA Control File

The following shows an FNA control file supplied as an instream data set in a batch SQL*Plus job. It could just as easily be a sequential disk data set or a member of a PDS.

```
//REPORT EXEC PGM=SQLPLUS, PARM='/nolog'
//STEPLIB DD DISP=SHR, DSN=ORACLE.V10G.CMDLOAD
//ORA$LIB DD DISP=SHR, DSN=ORACLE.V10G.MESG
//SQL DD DISP=SHR, DSN=YQB03.PROD.SQL <-- PDS containing SQL
//ORA$FNA DD *
/*****
 * Example FNA Control File *
 *****/

FSA(
  FTYPE(sql)
  FNAME('dd:sql(+)')
)

FSA(
  FTYPE(lst)
  FNAME(''YQB03.REPORTS.+''')
  FATTR(';space=(4096,(1000,200)),recfm=VBA,lrecl=137')
)
/*
//SYSIN DD ...
```

The example has FNA entries for the file extensions `.sql` and `.lst`. These extensions are associated with input SQL statement files and output SPOOL command files respectively. In the SQL*Plus input (not shown previously), a command which calls a file of SQL statements, such as the following:

```
get report2
```

This results in access to member REPORT2 in the partitioned data set YQB03.PROD.SQL. A SQL*Plus SPOOL command to direct output to a file, such as the following:

```
spool weekly
```

This results in writing the data set YQB03.REPORTS.WEEKLY. Assuming this data set does not already exist, it is created with a primary allocation of 1000 4K blocks and a secondary of 200 4K blocks. The data set is written with a record format of VBA and logical record length 137. The physical block size has been allowed to default.

Data Set DCB Attributes

Whenever Oracle opens an output filespec denoting a non-VSAM data set, DCB attributes must be determined. This is especially significant in the case of disk and tape data sets when the data set is to be processed or used by other software with specific DCB requirements, including non-Oracle software. These are situations where you may need to do something to ensure that specific DCB attributes are used. Multiple mechanisms are involved in establishing DCB attributes, including Oracle components such as FNA and z/OS components such as LE and DF/SMS. External factors such as the presence of DCB attributes on a DD statement or, for a preexisting data set, in the VTOC, also are involved.

Of chief importance in this area is LE, which performs the z/OS OPEN function for most data sets and thus controls much of what goes on. Even when you use FNA to request DCB attributes, Oracle simply passes the attribute keywords to the C function `fopen()` and LE does the work.

Default DCB Attributes

In the following discussions various conditions lead to the setting of default DCB attributes. This refers to LE defaults, which are documented in z/OS C/C++ Programming Guide. The LE defaults depend on whether the file usage is "text" or "binary" (which comes from the software) and on the type of device or data set being used. In virtually all situations relevant to end users, Oracle opens files as "text" rather than "binary."

The LE default DCB attributes for text files depend on the type of device as shown in the following table:

Device	RECFM	LRECL	BLKSIZE
disk	VB	1028	6144
SYSOUT	VBA	1028	6144
tape	VB	1028	32760

DCB Attributes from FNA

When you use an FNA entry with FATTR as discussed in an earlier section, DCB attribute keywords from the FATTR string are passed to LE when the file is opened. In

this case Oracle software takes no special action to influence what is finally established for DCB attributes; the outcome is simply normal LE behavior. The IBM manual *C/C++ Programming Guide* discusses some of the considerations in this situation, including the cases where not all three attributes (recfm, lrecl, and blksize) are specified. In general, the attributes you specify end up as the established attributes for the data set with any missing attributes defaulted as described in the IBM manual.

With sequential data sets, this process disregards any DCB attributes that might already exist (for a preexisting data set) or be coded in JCL (for a data set specified with DD:). In other words, DCB attributes from FNA will override both existing data set DCB attributes and DCB attributes you have specified on a DD statement or in a TSO ALLOCATE command. The issue arises only with sequential data sets. With an existing partitioned data set, if DCB attributes are specified in FNA they must match the established attributes of the data set.

To prevent changing existing sequential data set attributes, LE provides the special attribute notation `recfm=*`, which indicates that DCB attributes from the VTOC or on a DD statement are to be retained. When used, `recfm=*` should be coded without accompanying `lrecl=` or `blksize=` keywords. If these are included with `recfm=*` the values supplied must match the existing attributes.

A problem with `recfm=*` is that it eliminates LE default processing in determining DCB attributes. If you use this special attribute with a data set that does not have established DCB attributes, the data set ends up with the z/OS system default of `RECFM=U`. This is mostly likely to arise with a DD: filespec where the DD or allocation specifies `DISP=NEW` and no DCB attributes.

Most users do not intend to create data sets with a record format of U. Recently, IBM introduced another special attribute, `recfm=+`. This works the same as `recfm=*` except for the case where there are no preexisting attributes: in that situation, the LE defaults discussed in the previous section are used. Support for `recfm=+` was provided in IBM APAR PQ77259 which may or may not be installed on your system. Check with your z/OS system administrator to find out.

DCB Attributes without FNA

When Oracle opens a file for output and there is no FNA FATTR data associated with the type of file, the outcome depends on whether there are preexisting DCB attributes in the VTOC (for a disk data set) or on the DD statement or allocation (in the case of a DD: filespec). When there are preexisting attributes, they are retained as the real attributes for the data set provided they do not conflict with Oracle's intended usage. If there are no preexisting attributes, the LE defaults described earlier are used. This is essentially the behavior of LE's `recfm=+` described previously, but it does not require that the fix for IBM APAR PQ77259 be installed. The additional checking and defaulting logic is provided by Oracle software.

Interrupt Processing in TSO and z/OS UNIX Shell

When Oracle tools, utilities, or user-written applications run in an interactive environment (where there is a human user involved) it sometimes is desirable to interrupt a request that has been sent to the database server for processing. For example, when issuing ad hoc SQL from SQL*Plus, the user might realize that a SELECT is going to run too long in the server or produce too much output to view at the terminal. For these situations an interrupt mechanism is provided in the Oracle program interface code. The internals of the mechanism differ between TSO and a z/OS UNIX shell.

When an application running in TSO connects to an Oracle server, a subtask is automatically attached. The subtask uses the TSO STAX service to establish processing for TSO attention interrupts initiated with the 3270 ATTN or PA1 key. If you cause an attention interrupt while a request is running in the Oracle server, the message "Oracle ATTN." is displayed at the terminal and the server request is halted with an ORA-01013 error ("user requested cancel of current operation"). Halting of the server request is not necessarily instantaneous but usually occurs within a second or two.

In a shell, interrupt processing is implemented using POSIX signal processing that is part of z/OS UNIX System Services and LE. The internals are completely different from TSO and the interruption keying may differ (it normally is Ctl-C in a TTY session, for example) but the overall behavior is similar to that described for TSO. As in the TSO case, there may be a slight delay before the server recognizes the interrupt condition and terminates the current operation with ORA-01013.

Additional capabilities related to interrupt processing are provided in SQL*Plus. Refer to [Chapter 4, "Oracle Tools and Utilities on z/OS"](#) for details.

Oracle C Runtime Compatibility

Earlier to Oracle9i Release 2, Oracle used its own C runtime library instead of LE. Developed in the early 1980's before POSIX services and HFS files existed on MVS, it differs from LE in certain areas, particularly filespec syntax and related processing. In its LE implementation Oracle provides compatibility for certain Oracle C runtime conventions. This compatibility is enabled by default so that existing customer jobs, scripts, and other types of programs continue to work without change. Oracle recommends that new customers (and new applications developed by existing customers) disable this compatibility or at least avoid using Oracle C runtime syntax, which conflicts slightly with LE conventions.

Controlling Compatibility

Most of the compatibility feature is controlled by the environment variable `ORA_RTL_COMPAT`. In non-POSIX environments, when this variable is not set or is set to `ON`, compatibility is enabled. When it is set to `OFF`, compatibility is disabled. The compatibility feature is always disabled when `POSIX` is `ON`, regardless of this environment variable.

You can specify `ORA_RTL_COMPAT` using any of the non-program environment variable mechanisms discussed earlier in this chapter. Setting it programmatically for example, using a call to `C putenv()` is not supported. The variable must retain the same setting (or not be set at all) for the life of an application.

Your installation may choose to disable the compatibility feature by default using the global environment file feature described earlier in this chapter. In that case, to enable compatibility in an application, you must use a local (`ORA$ENV`) environment file with `ORA_RTL_COMPAT=ON` specified.

Because it runs before environment initialization, the compatibility feature for redirection operators is always enabled when `POSIX` is `ON` and is not affected by `ORA_RTL_COMPAT`.

Filespec Compatibility

When Oracle C runtime compatibility is enabled, filespecs that you supply in parameters, commands, and through FNA that use Oracle C runtime syntax are internally converted to equivalent LE syntax before they are used. Oracle C runtime

used a filespec prefix notation based on HFS syntax in which a leading directory was interpreted as a keyword indicating the type of filespec to follow. The list shows the Oracle C runtime prefixes (on the left) and the LE syntax (on the right) to which they are converted when compatibility is enabled:

```
/DD/name//DD:name
/DD/name(mem)//DD:name(mem)
/ODD/name// *DD:name
/ODD/name(mem)// *DD:name(mem)
/DSN/name// 'name'
/DSN/name(mem)// 'name(mem) '
/ODSN/name// * 'name'
/ODSN/name(mem)// * 'name(mem) '
```

The Oracle C runtime prefixes were case-insensitive, so filespecs with prefixes such as /dd/ or /ODsn/ are also converted, as shown in the prior list.

Note: Since the Oracle C prefix notation is valid HFS syntax in LE, enabling compatibility means you will be unable to access HFS files in directories with names like /DD and /dsn from a POSIX OFF application.

File Attribute Compatibility

Oracle C runtime also supported inclusion of JCL-like attribute information in a filespec (following a comma or full colon). The same attributes could also be supplied through the FATTR keyword in an FNA entry. When compatibility is enabled, a subset of the available Oracle C runtime attribute keywords are converted to LE equivalents. (These are the same LE attributes that you can use directly in FNA, discussed earlier.) Oracle C attributes without an LE equivalent are discarded. The following list shows the Oracle C attribute on the left, and the LE equivalent to which they are converted, on the right:

BLKsize=x	blksize=x
BLock=x	space=(x,...)
CYL	space=(cyl,...)
DIrectory=x	space=(, (, x))
F	recfm=F
FA	recfm=FA
FB	recfm=FB
FBA	recfm=FBA
FBAS	recfm=FBAS
FBS	recfm=FBS
LRecl=x	lrecl=x
PRIMary=x	space=(, (x,...))
RECFM=x	recfm=x
SECondary=x	space=(, (, x,...))
TRK	space=(trk,...)
V	recfm=V
VA	recfm=VA
VB	recfm=VB
VBA	recfm=VBA

Data Set Name Prefixing Compatibility

Besides the filespec and attribute conversions, enabling Oracle C runtime compatibility has another significant effect on file processing. With Oracle C runtime, prefixing of data set names that were not quoted or began with /DSN/ or /ODSN/ was done only in TSO, when a PROFILE PREFIX was set. LE prefixes unquoted data set

names in non-TSO applications, using the z/OS user ID associated with the address space. When Oracle C runtime compatibility is enabled, the prefixing normally done by LE in non-TSO environments is disabled in POSIX OFF applications.

An exception to Oracle C runtime compatibility occurs in file redirection operators when redirection specifies a z/OS data set name without a /DSN/ prefix or surrounding apostrophes. In a batch job, Oracle C runtime would not prefix such a data set name but in Oracle's LE implementation prefixing does occur. This behavior is not affected by the ORA_RTL_COMPAT environment variable.

Standard Files and Redirection Compatibility

Oracle C runtime used different defaults for two of the three standard files discussed earlier in this chapter and also supported redirection syntax that is not supported by LE. Compatibility in this area, which pertains only to Oracle tools and utilities running in POSIX OFF environments, is provided using the LE exit routine CEEBXITA. This exit runs before the C environment is initialized and is permitted to examine and modify parameters specified on the command line or in the PARM string. Because it runs before environment initialization, this processing is not affected by the ORA_RTL_COMPAT environment variable.

Since it cannot be controlled by the user, this compatibility processing is conditioned on detecting when Oracle C runtime conventions are being used. Detection is based on the use of Oracle C syntax for redirection (both the redirection operators and the following filespecs) and on the presence or absence of certain DD names.

The detection and compatibility processing proceeds as follows:

1. If the Oracle C standard error redirection operator "?" or "??" is found in the command line or PARM, it is converted to the LE equivalent "2>" or "2>>", respectively.
2. All redirection operators in the command line or PARM are examined for the use of Oracle C filespec syntax (/DD/, /ODD/, /DSN/, or /ODSN/). Any that use this syntax are changed to LE equivalents described in ["Filespec Compatibility"](#) on page 2-29.
3. If no redirection of standard error was found and a SYSERR DD statement is present, standard error is redirected to it using 2>//DD:SYSERR.
4. If there was no redirection of standard output and a SYSOUT DD statement is present, standard output is redirected to it using 1>//DD:SYSOUT. In a batch job, this occurs only if a SYSERR DD statement also is present.
5. In a batch job only, if any one or more of these conversions was done and there are redirection operators with data set name filespecs that do not use apostrophes, those filespecs are modified to enclose the data set name in apostrophes.

The effect of this somewhat complex set of rules is to permit both Oracle C runtime syntax and conventions and LE syntax and conventions to coexist as much as possible. One limitation of this implementation is that the standard files cannot be redirected to HFS files under directory names like /DD or /dsn because they are interpreted as Oracle C syntax.

Oracle Net and Server Connections on z/OS

Architecturally, the Oracle database server views all client connections as network connections regardless of whether the connecting client is on the same or a different computer than the server. When you run Oracle tools, utilities, or your own Oracle-accessing 3GL programs on z/OS, Oracle Net software that is part of the Oracle program interface code is executed—even if you are connecting to an Oracle instance on the same z/OS system using cross-memory services. This means Oracle Net features, parameter files, and so on, come into play in any Oracle programs you run on z/OS. This chapter describes z/OS-specific considerations in using Oracle Net and supplements the product information in the *Oracle Net Services Reference Guide*.

When we use the term "application" in this chapter we mean any Oracle-accessing program that you run on z/OS, including Oracle-supplied tools and utilities and your own 3GL applications using OCI or an Oracle Precompiler language.

Different Oracle Net considerations apply to Oracle-accessing programs running under Oracle Access Manager for CICS TS or Oracle Access Manager for IMS TM. Refer to [Chapter 6, "Developing Oracle Applications to Run on z/OS"](#) for more information.

This chapter contains the following sections:

- [Protocols](#)
- [Oracle Net Files](#)

Protocols

Oracle Net on z/OS supports two different protocols: cross-memory (XM) and TCP/IP (TCP). Normally you use XM protocol to connect to a local Oracle server (on the same z/OS system as the application) and TCP protocol to connect to a remote Oracle server (on z/OS or any other operating system on which Oracle runs). You can also use TCP protocol to connect to a local server, but it is less efficient than using XM. (There are, however, circumstances in which you might do this in order to exploit certain server features, discussed later.) You cannot use XM protocol to connect to an Oracle server on a different system than the application.

Most applications connect to just one Oracle server and so use only a single protocol at a time. Applications can connect to multiple Oracle servers, however, and it is possible for an application to mix protocols, such as using XM for one connection and TCP for another, simultaneously.

Regardless of which protocol you use, Oracle Net address data for a server must be available in order to locate and connect to the server. Address data depends on the protocol used and includes things like a TCP/IP hostname and port, server name or identifier. The address data for a server can be imbedded in the application program,

supplied through an external (but local) mechanism such as a parameter file or environment variable, or obtained from a remote name resolution service such as Oracle Names or an LDAP service. Using a remote name resolution service is applicable to TCP connections only; server address data for XM connections is always supplied locally.

Protocol and Address Determination

When an Oracle tool, utility, or 3GL application runs on z/OS, it makes a specific call to connect to an Oracle server. The details of this call vary depending on application design and on which program interface (API) is used, but in all cases the application is allowed to supply a character string representing either a name or address data for the target server. With Oracle tools and utilities this data usually is combined with the Oracle user ID and password into a single string supplied on the command line or on an input parameter or command. By convention, the user ID and password are separated by a forward slash and the password and name or address data are separated by an "at" sign (@). So, for example, a SQL*Plus session might be started in TSO with a command similar to the following:

```
READY
sqlplus scott/tiger@testdb1
```

In this example the user ID and password are `scott` and `tiger` respectively, and a target server name `testdb1` is included.

The behavior of Oracle Net in determining the protocol and address data depends on whether the connect call supplies target server data and whether that data specifies a name or an address:

- If server data is supplied and is a simple or dotted name, the name is looked up in a local `tnsnames.ora` parameter file or using a remote names server. The choice between a file and a names server is controlled by other Oracle Net parameters. (Refer to the section ["Oracle Net Output Files"](#) on page 3-8.)
- If server data is supplied and is an Oracle Net address (normally beginning with a left parenthesis), the address is used verbatim and no external file or remote names server is accessed.
- If server data is not supplied, a default hierarchy of external address data sources is examined to determine the protocol choice and target server address.

The external sources that are checked when server name or address data are omitted are, in descending order of precedence:

1. An included DD statement (or comparable TSO allocation) in the application job or session of the form `//ORA@sid DD DUMMY`, where `ORA@` is exactly as shown and `sid` is the 1-character to 4-character SID for an XM protocol address. Refer to the section ["XM Protocol Address"](#) on page 3-4. Only XM protocol is supported with this mechanism. The DD is not opened by Oracle software; it is supplied only to convey the SID. An Oracle SID on z/OS can be up to 8 characters long; SIDs longer than 4 characters cannot be accessed with this technique. If multiple DD names beginning with `ORA@` are present, the behavior is unpredictable.
2. A character value assigned to the environment variable `TWO_TASK`. This value is interpreted exactly the same as the target server data: it can be either a target server name (looked up in a `tnsnames.ora` file or remote name resolution service) or an Oracle Net address string. Either XM or TCP protocol can be specified with this mechanism.

3. A character value assigned to the environment variable ORACLE_SID. This value is interpreted as a 1-character to 8-character SID for an XM protocol address. Refer to "[XM Protocol Address](#)" on page 3-4. Lowercase letters in this SID are converted to upper case. Only XM protocol is supported with this mechanism.

If an application supplies no server name or address data on its connection attempt and none of the three mechanisms in the prior list is used, the connection will fail, typically with an ORA-06413 error.

XM Protocol

The XM protocol exploits the unique cross-memory hardware and software facilities of z/Series and z/OS. It can be used only when the application and the target Oracle server are on the same z/OS system. It cannot be used between separate nodes of a Sysplex cluster, even if the nodes are logical partitions (LPARs) in the same physical processor complex.

Unlike other protocols, XM does not use shadow or surrogate processes to execute application requests in the database server. Instead, the Oracle program interface issues a System/390 Program Call (PC) hardware instruction to transfer execution directly into the server address space to execute requests. Request input and output data are copied between the application and the server address spaces using cross-memory instructions. On conclusion of a server request, control returns to the application through a Program Return (PR) instruction. Thus, client/server interactions with XM protocol take place at processor instruction speeds, without expensive context-switching or task-switching operations.

There are several characteristics of the XM protocol of which you should be aware:

- All processor (CPU) time consumed by your application while executing in the Oracle server address space is accounted by z/OS under your application's task and address space, not those of the Oracle server. This CPU time appears, for example, in the z/OS SMF accounting data written for your application job or session, added to the CPU time that your application consumed on its own, locally.
- Database and other file I/O operations that your application issues while executing in the Oracle server address space are accounted to the Oracle server address space, not to your application job or session.
- When you use XM protocol, your database requests run at the z/OS dispatching priority or WLM goal of your job or session, not at the priority or WLM goal of the Oracle server address space.
- If your application uses Oracle server parallel execution features, such as parallel query processing, some of the application's work is done by local subtasks in the server address space. This work does run at the z/OS dispatching priority or WLM goal of the server address space and is not accounted by z/OS to your application job or session.
- Oracle for z/OS provides the ability to impose native z/OS security checks (through IBM's RACF or a comparable product) on XM protocol connections. If your installation has activated this feature, the z/OS user ID under which your application runs must be granted access authorization for the target Oracle server. This check is completely separate from Oracle logon processing, the Oracle user ID and password, and so on.
- Unlike TCP, the XM protocol is always, by its nature, synchronous and "half duplex." A few rather specialized server features require a protocol capable of asynchronous or nonblocking operation. To exploit those specific features on

z/OS you must use TCP protocol, even if the application and server are on the same z/OS system.

XM Protocol Address

If you are not using an `ORA@sid` DD statement or an `ORACLE_SID` environment variable to indicate use of XM protocol, you will need to code an XM protocol address, either in a `tnsnames.ora` file, discussed in "[Oracle Net Files](#)" on page 3-6, or directly in a connect parameter or argument. The generic considerations for coding protocol addresses are described in the *Oracle Net Services Reference Guide*. Refer to the guide for general syntax conventions and for details and examples illustrating TCP protocol. Here we describe the name-value pairs that make up an XM protocol address, unique to z/OS. All pairs described are used in the ADDRESS portion of the protocol address.

```
(PROTOCOL=xm)
```

This pair is specified exactly as shown and is required to indicate that XM protocol is being used.

```
(SID=sid)
```

This specifies the 1-character to 8-character SID (Service Identifier) of the target z/OS Oracle server. The SID is a unique identifier assigned to each Oracle server configured on a given z/OS system. All SIDs use upper case letters; any lowercase letters are converted to upper case. If you specify SID, which is recommended, neither of the following two parameters is used. If you do not specify SID, both of the following parameters are required.

```
(SUBSYS=ssn)
```

This specifies the 1-character to 4-character z/OS subsystem name of the OSDI subsystem in which the target Oracle service is defined. Any lowercase letters in `ssn` are converted to upper case.

```
(SERVICE=svname)
```

This specifies the 1-character to 8-character OSDI service name of the target Oracle server. Any lowercase letters in `svname` are converted to upper case.

XM Protocol Examples

Here we show several examples that use XM protocol. All use SQL*Plus, but the techniques shown extend readily to other tools and utilities and to your own 3GL applications.

In the first example, SQL*Plus is invoked at the TSO READY prompt and the XM address is passed directly in the first command line parameter with the user ID and password. The target server has the SID "ODB1". This example assumes that `TSOLIB/STEPLIB` and `ORA$LIB` are already allocated as needed.

```
READY
sqlplus scott/tiger@(address=(protocol=xm)(sid=odb1))
```

In the next example, a batch job execution of SQL*Plus omits the server data from the user ID/password parameter and uses an `ORA@sid` DD to specify the same target as the prior example.

```
//PLUS EXEC PGM=SQLPLUS, PARM='SCOTT/TIGER'
//STEPLIB DD DISP=SHR, DSN=ORACLE.V10.COMDLOAD
//ORA$LIB DD DISP=SHR, DSN=ORACLE.V10.MESG
//ORA@ODB1 DD DUMMY
```

```
//SYSIN DD DISP=SHR,DSN=FDNR30.PROD.SQL(WNP14C)
//
```

In the third example, a similar batch job relies on an entry in a `tnsnames.ora` file to obtain the XM address data. The `tnsnames.ora` file is discussed in "Oracle Net Files" on page 3-6. Typically this file would be a data set or PDS member rather than instream as shown in the following example:

```
//PLUS EXEC PGM=SQLPLUS,PARM='SCOTT/TIGER@PRODDb1'
//STEPLIB DD DISP=SHR,DSN=ORACLE.V10.COMDLOAD
//ORA$LIB DD DISP=SHR,DSN=ORACLE.V10.MESG
//TNSNAMES DD *
proddb1 =
  (description=
    (address=(protocol=xm)(sid=odbl)))
/*
//SYSIN DD DISP=SHR,DSN=FDNR30.PROD.SQL(WNP14C)
//
```

The last example shows the use of an `ORACLE_SID` environment variable in a z/OS UNIX shell to specify the target server. It assumes that the required `ORACLE_HOME`, `PATH`, and `LIBPATH` are already set.

```
$ export ORACLE_SID=odbl
$ sqlplus scott/tiger
```

TCP Protocol

Oracle Net TCP protocol provides connectivity between client and server over a LAN or WAN as well as local connections. All Oracle platforms implement TCP/IP protocol, so it can be used by outbound z/OS client applications to connect to Oracle servers on any supported platform. If the installation configures and runs an Oracle Net (Listener) service on z/OS, inbound remote clients on any platform can connect to Oracle servers running on z/OS. This material deals mainly with outbound z/OS client considerations; inbound client considerations are covered in the *Oracle Database System Administration Guide for IBM z/OS on System z*.

The following considerations pertain to outbound client use of TCP protocol on z/OS:

- In Oracle9i Version 9.0 and earlier, outbound z/OS clients' network I/O operations were performed by the Oracle Net address space. Thus, clients were dependent on the Net service being up and operational, and an interruption in the Net service impacted client operations. Starting with Oracle 9i Version 9.2 and in all subsequent releases, outbound clients interact directly with the z/OS TCP/IP service and are not affected by Oracle Net service availability.
- IBM's z/OS TCP/IP protocol service can be stopped and started. If it is stopped, in-progress Oracle Net TCP connections receive permanent errors and all new connection attempts fail until the service is restarted.
- Using z/OS TCP/IP requires the application to be a z/OS UNIX System Services process. If the application is running in a z/OS UNIX shell, it already is a POSIX process and the requirement is met. Batch jobs and TSO sessions (not in a shell) generally are not z/OS UNIX processes and the first use of TCP/IP (or any other z/OS UNIX interaction) will cause the job or session to be "dubbed" or made a z/OS UNIX process.

For dubbing to succeed, the z/OS user ID associated with the job or session must have a default OE segment defined to the z/OS security service. If dubbing fails, TCP/IP interactions fail and the client will be unable to connect to a server using

TCP. Successful dubbing does not affect application program operations nor does it change the POSIX indicator. If you are uncertain about whether your application's z/OS user ID is capable of being dubbed, consult with your security administrator.

- Using TCP protocol means it is possible to use a remote name resolution service such as Oracle Names or LDAP instead of locally-configured address data (for example, a `tnsnames.ora` file) to contact Oracle database servers. Consult with your systems administrator or database administrator to determine if name resolution services are available in your installation. Using a remote name service will also require dubbing independent of the protocol selected as a result of the lookup.

Oracle Net TCP protocol address particulars are covered in the *Oracle Net Services Reference Guide*.

Oracle Net Files

When you use Oracle Net (including local server connections) any of several Oracle Net input (parameter) files may be involved. In some situations output files (log or trace files) may also be produced. General considerations for these files and what goes in them are covered in the *Oracle Net Services Reference Guide*. Here we address issues specific to z/OS.

A primary consideration is how each file is identified. On z/OS, Net default filespecs differ depending on the POSIX indicator. Remember that with Oracle tools and utilities, POSIX is determined by whether the program is initiated from a z/OS UNIX shell or not. (POSIX is ON in a z/OS UNIX shell, OFF otherwise.) In your own 3GL applications, the POSIX setting can be built into the application module, specified at runtime, or set automatically based on where the application runs (as with Oracle tools and utilities).

Oracle Net input (parameter) files are accessed when an application makes its first attempt to connect to an Oracle server. In some cases, they may be opened and read multiple times in the life of an application. The following sections correspond to similarly-titled sections in the *Oracle Net Services Reference Guide*.

Profile Parameters (`sqlnet.ora`)

This is the primary parameter file of Oracle Net, usually referred to by its UNIX HFS file name `sqlnet.ora`. It is optional, but must be supplied if anything other than default parameters is desired. Activating Oracle Net logging or tracing, using a remote name resolution service, and other Oracle Net features require this file to be supplied.

In a POSIX ON environment, this file has the name `sqlnet.ora` and is located in one of several HFS directories as described for UNIX systems in the *Oracle Net Services Reference Guide*.

When POSIX is OFF, this file is opened as `//DD:SQLNET` which means it is specified by the SQLNET DD statement. In this case the file can be a sequential data set, a PDS member, an instream (DD *) data set, or an HFS file (DD PATH=). A sequential or partitioned data set can have record format F, FB, V, or VB. Data that is not a part of the Oracle Net input, such as sequence numbers, is not permitted.

Some of the parameters you might specify in `sqlnet.ora` have z/OS-specific considerations, as follows:

LOG_DIRECTORY_CLIENT

This parameter indicates the "directory" in which Oracle Net log files are written when client logging is activated. On z/OS, it can supply an HFS directory path or the left-hand portion of a data set name, as discussed in [Chapter 2, "Oracle Software Interaction with z/OS and Language Environment."](#) If this parameter is specified, it must be syntactically compatible with whatever is specified or defaulted for LOG_FILE_CLIENT. If LOG_FILE_CLIENT is a filespec to which no directory can be added, this parameter is ignored.

The default for this parameter is an empty string, signifying the current working directory (POSIX ON) or default LE data set name prefixing (POSIX OFF). Note that when OraRTL compatibility is enabled, no data set name prefix is added in non-TSO environments.

LOG_FILE_CLIENT

This parameter specifies either a complete or trailing (right hand) portion of the filespec for Oracle Net logging. This may be combined with the LOG_DIRECTORY_CLIENT value to produce the complete filespec. If this parameter specifies an absolute HFS path or a non-HFS filespec that cannot be modified (such as a //DD: or //SYSOUT: filespec), the LOG_DIRECTORY_CLIENT part is not incorporated. If the two strings are combined, the result must be a valid, usable LE filespec.

The default for this parameter is `sqlnet.log` when POSIX is ON and `//DD:SQLNETLG` when POSIX is OFF. Note that the POSIX OFF default means any setting of LOG_DIRECTORY_CLIENT is ignored.

TRACE_DIRECTORY_CLIENT

This parameter indicates the "directory" in which Oracle Net trace files are written when client tracing is activated. On z/OS, it can supply an HFS directory path or the left-hand portion of a data set name as discussed in [Chapter 2](#). If this parameter is specified, it must be unambiguous and compatible with whatever is specified or defaulted for TRACE_FILE_CLIENT. If TRACE_FILE_CLIENT is a filespec to which no directory can be added, this parameter is ignored.

The default for this parameter is an empty string, signifying the current working directory (POSIX ON) or default LE data set name prefixing (POSIX OFF). Note that when OraRTL compatibility is enabled, no data set name prefix is added in non-TSO environments.

TRACE_FILE_CLIENT

This parameter specifies either a complete or trailing (right hand) portion of the filespec for Oracle Net tracing. This may be combined with the TRACE_DIRECTORY_CLIENT value to produce the complete filespec. If this parameter specifies an absolute HFS path or a non-HFS filespec that cannot be modified (such as a //DD: or //SYSOUT: filespec), the TRACE_DIRECTORY_CLIENT part is not incorporated. If the two strings are combined, the result must be a valid, usable LE filespec.

The default for this parameter is `sqlnet.trc` when POSIX is ON and `//DD:CLITC` when POSIX is OFF. Note that the POSIX OFF default means any setting of TRACE_DIRECTORY_CLIENT is ignored.

TNSPING.TRACE_DIRECTORY

This parameter specifies the directory in which the TNSPING trace file is written. The default is `$ORACLE_HOME/network/trace` when POSIX is ON. This parameter is ignored when POSIX is OFF.

Trace information is written to the `tnsping.trc` file when POSIX is ON and to `//DD:TNSPING` when POSIX is OFF.

Local Naming Parameters (`tnsnames.ora`)

This file supplies Oracle server name resolution mappings. When the server data supplied on a connection attempt is a name (rather than an explicit Oracle Net address string) and you have specified (or defaulted) "local naming" in `sqlnet.ora`, this file is searched for an entry matching the supplied name. The file is optional, but if you omit it and attempt to connect to a server identified by name, and have not enabled an alternative name resolution method (such as a remote resolution service), the connection attempt will fail.

In a POSIX ON environment, this file has the name `tnsnames.ora` and is located in one of several HFS directories as described for UNIX systems in the *Oracle Net Services Reference Guide*.

When POSIX is OFF, this file is opened as `//DD:TNSNAMES` which means it is specified by the TNSNAMES DD statement. In this case the file can be a sequential data set, a PDS member, an instream (DD *) data set, or an HFS file (DD PATH=). A sequential or partitioned data set can have record format F, FB, V, or VB. Data that is not a part of the Oracle Net input, such as sequence numbers, is not permitted

Except for the XM protocol address particulars, already discussed, there are no z/OS-specific considerations for the data in this file.

Directory Usage Parameters (`ldap.ora`)

This file supplies configuration parameters related to using directory-based server name resolution. You may need to supply it when your `sqlnet.ora` NAMES.DIRECTORY_PATH parameter includes LDAP as a naming method.

In a POSIX ON environment, this file has the name `ldap.ora` and is located in one of several HFS directories as described for UNIX systems in the *Oracle Net Services Reference Guide*.

When POSIX is OFF, this file is opened as `//DD:LDAP` which means it is specified by the LDAP DD statement. In this case the file can be a sequential data set, a PDS member, an instream (DD *) data set, or an HFS file (DD PATH=). A sequential or partitioned data set can have record format F, FB, V, or VB. Data that is not a part of the Oracle Net input, such as sequence numbers, is not permitted

There are no z/OS-specific considerations for the data in this file.

Oracle Net Output Files

When requested through `sqlnet.ora` parameters, Oracle Net writes log and trace files. Usually this occurs during problem diagnosis, at the request of Oracle Support Services. The type and name (filespec) for these files is controlled by the four parameters discussed under "[Profile Parameters \(sqlnet.ora\)](#)" on page 3-6. If these parameters are not specified, the log and trace files default to `//DD:SQLNETLG` and `//DD:CLITC` in POSIX OFF applications and to `sqlnet.log` and `sqlnet.trc` (in the current working directory) in POSIX ON applications.

Oracle Tools and Utilities on z/OS

This chapter discusses z/OS-specific issues and considerations with various Oracle tool and utility programs that application developers are likely to use. (Tools used chiefly by database administrators or system administrators, such as RMAN, are covered in the *Oracle Database System Administration Guide for IBM z/OS on System z*. Before using this chapter you should be familiar with the material in [Chapter 2, "Oracle Software Interaction with z/OS and Language Environment,"](#) which describes features and considerations common to most or all Oracle tools and utilities. You should also have access to the base or generic documentation for each tool or utility you are using since the material in this chapter is supplemental to that.

Oracle supplies JCL procedures for some of the tools and utilities that support batch job execution. Consult with your system administrator to determine if these procedures are installed in your system and the procedure library and member names used.

This chapter contains the following sections:

- [SQL*Plus](#)
- [SQL*Loader](#)
- [Export and Import](#)
- [Datapump Export and Import](#)
- [TKPROF](#)

SQL*Plus

SQL*Plus is a general purpose tool for issuing both prepared and ad hoc SQL statements and PL/SQL procedure calls. It also is used for database administration and operations functions, including creating database objects and structures, managing Oracle user IDs and object security, and starting up and shutting down database instances. When you run SQL*Plus on z/OS you can interact with Oracle servers on z/OS or on any other Oracle platform. Refer to the *SQL*Plus User's Guide and Reference* for complete generic information about SQL*Plus.

Invocation

SQL*Plus on z/OS supports batch job, TSO, and z/OS UNIX shell execution environments. A JCL procedure for batch execution (default name ORASQL) 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 the non-POSIX environments, the load module or program object name is SQLPLUS.

In a z/OS UNIX shell (including OMVS under TSO), use the `sqlplus` (lowercase) command to invoke this tool.

SQL*Plus reads C standard input (described in [Chapter 2](#)) as the primary source of input commands and SQL statements. If you do not supply an Oracle user ID/password (or the special value `/nolog`) as an invocation parameter, SQL*Plus prompts for a user ID and password and reads those from standard input before reading for normal commands. If you do this in a batch job, the prompt responses must appear in the first records or lines of standard input, ahead of any input commands or SQL statements. In interactive usage (TSO or a shell), these prompts and responses normally occur at the terminal. In this case, if the password is entered separately from the user ID, it is not displayed at the terminal.

For certain database control operations (such as Oracle startup and shutdown), SQL*Plus requires a special user ID/password parameter that contains spaces, such as `/ AS SYSDBA.` Refer to the section "[Parameters Containing Spaces](#)" on page 2-9 for information about how to specify such values.

SQL*Plus output (including the aforementioned prompts) is written to C standard output. There are various options to control exactly what gets written as well as a SPOOL command to write output to a separate data set or file.

Output Interruption

Interruption refers to using the terminal ATTN or PA1 key (in TSO) or a POSIX signal mechanism such as Ctl-C to interrupt in progress activity. Besides the normal interrupt handler that is established when connecting to Oracle from TSO or a shell, SQL*Plus establishes its own handler to allow termination of display output. If you generate an interrupt while using SQL*Plus, the behavior depends on whether SQL*Plus or the target Oracle server was in control when the interrupt was processed. If the Oracle server was in control (it was processing a SQL statement or PL/SQL procedure call from SQL*Plus) the interrupt halts the in progress statement with an ORA-01013 error, as described in [Chapter 2](#). If the interrupt occurs while SQL*Plus is in control, driving the SQL*Plus-specific handler, it causes current fetches for any SELECT statement to stop. The typical occurrence of the latter situation is where you have issued a SQL SELECT statement that produces more output than expected. This provides an alternative to canceling or killing the entire SQL*Plus session to halt unwanted output.

Profile Files

SQL*Plus reads optional profile files during initialization (after logging on to Oracle). The files can contain SQL*Plus commands (such as SET commands) and SQL statements and PL/SQL blocks.

When SQL*Plus runs in a shell on z/OS, there are two profile files with the same names and locations as on other UNIX platforms. The site profile applies to all users and is read from `$ORACLE_HOME/sqlplus/admin/glogin.sql`. The user profile is read from `login.sql` in the first HFS directory (of multiple possible directories) in which a file of that name is found. The directories that are searched for the user profile are the current working directory followed by any directories listed in the SQLPATH environment variable. (Refer to the *SQL*Plus User's Guide and Reference* for details on using the SQLPATH environment variable.)

In z/OS batch and TSO (POSIX OFF), there is no site profile file. The user profile filespec is `//DD:SQLLOGIN` which means the SQLLOGIN DD is read for profile statements. The DD can specify any of the z/OS data set or file types permitted for SQL files, discussed in the section "[SQL Files](#)" on page 4-3.

Profile files are optional. If a file cannot be opened, profile processing is skipped and no error message is issued.

SQL Files

Various SQL*Plus commands operate on files containing SQL, PL/SQL, and/or SQL*Plus commands. On z/OS, these files can be a sequential data set, a PDS member, an instream (DD *) data set, or an HFS file. A sequential or partitioned data set can have a record format of F, FB, V, or VB. Records in a data set (including instream) must not contain sequence numbers or other data not intended for SQL*Plus to read.

If the file you want to use is counter to the POSIX expectation (for example, an HFS file when POSIX is OFF), you must use an unambiguous filespec when specifying the file. When a file is specified as a data set name or HFS file name, the name is subject to extension processing (as discussed in [Chapter 2](#)) with the type suffix `sql`. The SQL*Plus commands `@`, `@@`, `EDIT`, `GET`, `SAVE`, and `START` all apply this extension to supplied filespecs.

In POSIX OFF environments, you can use FNA to manipulate data set filespecs produced by name extension or to associate disk space and DCB attributes with files that are written. Refer to [Chapter 2](#) for a complete description of FNA. Use of FNA is required when you run Oracle catalog creation or upgrade scripts using SQL scripts that are members of a partitioned data set. In this case, FNA is used to transform simple 1-segment script names into PDS member names inside parentheses following a DD or data set name.

The SQL*Plus `@@` operator requires additional explanation on z/OS. This operator is shorthand for the `START` command, similar to `@` but with a slight difference: when an `@@` command is coded in a file and the filespec supplied on the command is not absolute; the file is read from the same directory that contains the file in which the `@@` command was coded. With data sets, "directory" refers to the high-level (leftmost) data set name qualifier, so the behavior of `@@` with data sets might not be what you expected. In particular, you cannot code `@@foo` within a PDS member and assume that `foo` will be treated as another member name within that PDS. Without FNA, `@@foo` will end up trying to access a data set named `hlq.FOO.SQL` where `hlq` is the high-level dsname qualifier of the data set in which the `@@foo` command is supplied. To get different behavior, such as treating `foo` as a member name, you must use FNA.

EDIT Command Processing

The SQL*Plus `EDIT` command is designed to invoke an external editor of your choosing. In z/OS TSO (POSIX OFF), the only external editor supported is IBM's ISPF editor. In this environment, the SQL*Plus variable `_EDITOR` is automatically set to `ISPF`.

To be able to invoke the ISPF editor in TSO, SQL*Plus requires that the ISPF environment already be established. To accomplish this, invoke SQL*Plus from within an ISPF session (from the ISPF "Command Shell" panel). Keep in mind that a data set name filespec supplied on an `EDIT` command is subject to extension (and FNA) processing. HFS files cannot be edited with the ISPF editor using SQL*Plus `EDIT`.

In a z/OS UNIX shell, `_EDITOR` defaults to `vi`. To use a different editor, define the `_EDITOR` variable to the exact name of the editor executable or shell script. Whether the editor you use in a shell is capable of processing data sets (in addition to HFS files) depends on the editor. Editing data sets with the default z/OS UNIX editor (`vi`) is not supported.

Neither vi nor ISPF work in the TSO OMVS shell. To use EDIT when running SQL*Plus in OMVS, you must define `_EDITOR` to `oedit` before issuing the EDIT command, as follows:

```
SQL> def _editor=oedit
SQL>
===> edit /tmp/load_it.sql
```

The editor, `oedit`, is a form of the ISPF editor that works in POSIX environments. It is the only editor supported when SQL*Plus runs in the OMVS shell. Only HFS files can be accessed with `oedit`.

The EDIT command is not available when SQL*Plus runs as a z/OS batch job.

SPOOL Command Processing

The SPOOL command causes SQL*Plus to begin writing output (including command responses and SQL statement results) to a specified file. On z/OS, the filespec you supply on the spool command can specify a sequential data set, PDS member, SYSOUT, or an HFS file. If the file you want to use is counter to the POSIX expectation, you must use an unambiguous filespec. If the filespec you supply is a data set or HFS file name, it is subject to extension with the suffix first. When running SQL*Plus in batch or TSO, you can use FNA to manipulate the resulting filespec or to associate space or DCB attributes with the data set.

If you spool to a SYSOUT filespec, as in the following:

```
SQL>
spool //s:*, ,MCMDEV3
SQL>
```

...

the output is freed (becomes eligible for routing or printing) as soon as the file is closed. The spool file is closed when you issue another SPOOL command (including the special command SPOOL OFF) or when you exit SQL*Plus.

Note: The SQL*Plus `LINESIZE` system variable must be less than or equal to the SPOOL dataset record length to avoid record truncation. If SQL*Plus `LINESIZE` exceeds `LRECL`, then error SP2-0308: cannot close spool file is thrown. The underlying LE error, O/S Message: EDC5003I Truncation of a record occurred during an I/O operation, can be trapped using the `WHENEVER OSERROR` command.

The special command SPOOL OUT is not supported on z/OS. Attempting to use it will produce an error message.

HOST Command Processing

The HOST command allows you to invoke an arbitrary external program or script from within a SQL*Plus session. This feature is implemented on z/OS using the `C` function `system`, described in the IBM manual *C/C++ Run-Time Library Reference*. Refer to this manual for details on the capabilities and limitations of the `system` function.

The behavior of `system()` (and thus of the HOST command) depends on the POSIX indicator and, in the POSIX OFF case, on command syntax and whether the environment is batch or TSO. In TSO, HOST can be used to invoke TSO commands,

CLISTs, and REXX EXECs. To do this, code the command text after HOST exactly as it would be entered at the TSO READY prompt, as in the following example:

```
SQL>
host listd misc.sql
JSMITH.MISC.SQL
--RECFM=LRECL-BLKSIZE=DSORG
  VB   255   4096   PO
--VOLUMES--
  PNW042
SQL>
```

In addition, in both TSO and batch, HOST can be used to invoke jobstep z/OS programs such as IBM utilities. To do this, special syntax (similar to the JCL EXEC statement) is required to indicate the program name and, optionally, PARM data in the command text as illustrated here:

```
SQL>
host pgm=iebcopy,param=COPY
IEBCOPY MESSAGES AND CONTROL STATEMENTS                PAGE    1
IEB1035I JSMITH  $TS1  $TS1 13:59:38 THU 02 OCT 2003 PARM='COPY'
IEB1099I *** IEBCOPY IS NOT APF AUTHORIZED ***
$TS1  COPY          INDD=SYSUT1,OUTDD=SYSUT2          GENERATED STATEMENT
...
```

With this technique, letter case is preserved in any PARM data that you supply, which is why COPY is in upper case in the example. Refer to the IBM manual *C/C++ Run-Time Library Reference* for additional details on this command syntax.

In non-POSIX TSO and batch environments, you cannot use SQL*Plus HOST to invoke an HFS executable or shell script. Also, you cannot issue HOST alone (with no command text) to be placed at a general command prompt.

The `system()` function and HOST behave differently when you run SQL*Plus in a z/OS UNIX shell. The command text that you supply with HOST must resolve to a valid executable or shell script file in the HFS, or be a built-in shell command. Commands are processed with the shell indicated by the SHELL environment variable; if SHELL is not set, the default shell `/bin/sh` is used. Be aware that commands issued through HOST are executing in a subshell relative to the shell in which SQL*Plus was invoked, so issuing commands that change the state of the current shell has no effect on the SQL*Plus session. For example, using HOST to issue a `cd` (change directory) command or to set an environment variable does not change the current working directory or the value of that environment variable as seen by SQL*Plus.

When SQL*Plus runs in a z/OS UNIX shell, HOST can be issued alone, without command text, causing the user to be placed at a general shell prompt. In this situation multiple commands can be issued in succession. Control returns to SQL*Plus when you end execution of the nested shell. For example:

```
SQL> host
$ cd /oradev/js1/src
$ ls -l *.c
-rw-rw-r-- 1 JSMITH  PDRP88      7438 Nov  5 15:55 ocidemo3.c
$ rm ocidemo3.c
$ exit
SQL>
```

In this case, the `cd` command is effective in terms of setting the directory used by the subsequent commands within the subshell.

There is no direct way to invoke non-HFS z/OS programs or TSO commands using SQL*Plus HOST in a POSIX ON environment.

TIMING Processing

The SQL*Plus timing feature, activated with the command SET TIMING ON, reports elapsed time for server SQL and PL/SQL operations. On z/OS, the timing feature also reports the processor (CPU) time consumed by SQL*Plus during these operations. As described in [Chapter 3](#), when you use XM protocol to connect to a local Oracle server, this time figure includes the Oracle server processing, which is executed by the SQL*Plus task in cross-memory mode. This allows the timing feature to be used to collect approximate server processing times for SQL and PL/SQL operations.

Be aware that server caching of data, SQL, and PL/SQL, and other workload-related factors, mean that processor time for a particular operation can vary significantly over multiple trials. Also, the processor time reported by the timing feature on z/OS does not include time consumed by server "slave" tasks used in parallel query and other parallel operations. Finally, when using TCP protocol rather than XM, the processor time reported by the timing feature does not include any Oracle server processor time.

Return Codes

Except for initialization failures, SQL*Plus normally sets a SUCCESS (zero) return code regardless of the success or failure of any SQL, PL/SQL, or SQL*Plus commands processed. To change this behavior, for example to set a nonzero return code if any SQL operation fails, you must use the EXIT command, possibly in conjunction with the WHENEVER command. EXIT allows you to set any valid numerical return code you want using either a specific number or the current value of a SQL*Plus variable. Remember, however, that in non-POSIX environments return codes are limited to the range 0-4095. If you issue EXIT with a number higher than 4095, the resulting return code is unpredictable.

You can also issue EXIT with one of three keywords whose numerical value depends on the platform. On z/OS, the values associated with these keywords are shown in the following table:

Keyword	Value
SUCCESS	0 (the default when EXIT has no parameter)
WARNING	4
FAILURE	8

On an initialization failure, SQL*Plus ends as though EXIT FAILURE occurred, producing a return code 8 on z/OS.

Refer to the *Oracle Database SQL*Plus User's Guide and Reference* for complete details on the EXIT and WHENEVER commands.

Unsupported Features

The following SQL*Plus features or commands are not available on z/OS:

- Site profile file, when running in POSIX OFF environments.
- SPOOL OUT command. This command results in an error message on z/OS.

- SET NEWPAGE 0 command. This command is accepted but does not clear the terminal screen (as it does on some ports) when issued in TSO or a shell.
- RUNFORM command. This command results in an error message on z/OS.

SQL*Plus Examples

The following example is SQL*Plus executed as a batch job step with both the user profile file and standard input provided as instream data sets. An Oracle database instance with SID ORA1 is accessed using cross-memory (XM) protocol, with the connection specified using an ORA@ DD statement.

```
//PLUS      EXEC PGM=SQLPLUS, PARM='/nolog', REGION=0M
//STEPLIB DD DISP=SHR, DSN=ORACLE.V10G.CMDLOAD
//ORA$LIB DD DISP=SHR, DSN=ORACLE.V10G.MESG
//ORA@ORA1 DD DUMMY
//SQLLOGIN DD *
whenever oserror exit 16
whenever sqlerror exit 12
set pagesize 0
/*
//SYSIN DD *
connect conklin/grommet6
create table big_acct (
  acct number,
  balance number,
  desc varchar2(200))
as
  select account_primary, balance_fwd, account_pdesc
  from pmds011.master
  where (balance_fwd > 100000.00) and
  (account_pdesc not null);
commit;
select * from sub_acct1 order by balance descending;
exit
/*
```

The following example shows SQL*Plus used in line-mode TSO and assumes the following:

- The CMDLOAD data set is provided as STEPLIB, TSOLIB, or a linklist library.
- The MESG data set is already allocated to ORA\$LIB.
- A TNSNAMES DD is allocated and contains a valid entry with the identifier ORALNX1. This is a TCP/IP connection to an Oracle 10g instance running on a z/Linux system.

```
READY
sqlplus pabr/asdfghj@oralnx1

SQL*Plus: Release 10.2.0.5.0 - Production on Fri Feb 3 01:17:27 2012
Copyright (c) 1982, 2010, Oracle. All rights reserved.

Connected to:
Oracle Database 10g Enterprise Edition Release 10.2.0.5.0 - Production
With the Partitioning, Oracle Label Security and Data Mining options

SQL>
host listd test.sql mem [Text entered by user]
host listd test.sql mem
```

```

JCONKLI.TEST.SQL
--RECFM-LRECL-BLKSIZE-DSORG
  VB   255   27998   PO
--VOLUMES--
  DSM119
--MEMBERS--
  RVNT030
  RVNT040
SQL>
get test(rvnt030)          [Text entered by user]
get test(rvnt030)
  1  SELECT DNAME, DACCT, DDESC FROM ANWR.MASTER WHERE
  2  (DLCODE IN (12, 43, 663, 900)) AND
  3* (DDESC NOT NULL);
SQL>
l 1          [Text entered by user]
l 1
  1* SELECT DNAME, DACCT, DDESC FROM ANWR.MASTER WHERE
SQL>
c /ANWR./TEMP./          [Text entered by user]
c /ANWR./TEMP./
  1* SELECT DNAME, DACCT, DDESC FROM TEMP.MASTER WHERE
SQL>
/          [Text entered by user]
/

          [SELECT output omitted]

SQL>
exit          [Text entered by user]
exit
Disconnected from Oracle Database 10g Enterprise Edition Release 10.2.0.5.0 -
  Production
With the Partitioning, Oracle Label Security and Data Mining options
READY

```

SQL*Loader

SQL*Loader is an Oracle utility that loads external data into Oracle database tables. When you run SQL*Loader on z/OS, the external data can come from z/OS data sets (including VSAM) and HFS files, and can include character, zoned decimal, packed decimal, binary integer, and hexadecimal floating point (HFP or System/370 floating point) data. SQL*Loader on z/OS can load data into an Oracle database on z/OS or on any other Oracle platform. Refer to *Oracle Database Utilities 10g Release 2 (10.2)* for complete generic information about SQL*Loader.

Invocation

SQL*Loader on z/OS supports batch job, TSO, and shell execution environments. A JCL procedure for batch execution (default name ORALDR) 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 the non-POSIX environments, the load module or program object name is SQLLDR. In a shell (including OMVS under TSO), use the `sqlldr` (lowercase) command to invoke this utility.

Unlike most tools and utilities, SQL*Loader does not read C standard input as a primary source of input. Loader processing details are specified in a text control file

whose filespec usually is supplied on the command line or PARM field. If you do not supply this parameter but you do supply a user ID/password parameter, Loader prompts for the control filespec and reads it from C standard input. This is intended mainly for interactive (TSO or shell) execution; if you do this in a batch job, the prompt response must be the first record or line of standard input. If you supply neither a control file nor a user ID/password as invocation parameters, SQL*Loader displays a help message and ends normally.

The parser used by SQL*Loader for command line parameters has its own syntax conventions (described in the generic documentation) and is sensitive to apostrophes (single quotes). If you use apostrophes in a command line filespec parameter, you must "escape" them with a preceding backslash to keep the parser from trying to interpret them as syntax elements. For example, to specify the CONTROL= parameter with the data set name filespec `// 'DMWONG.ORA.CTL(LOAD2) '` in a batch job, code

```
//LOAD EXEC PGM=SQLLDR,PARM='control=//\'DMWONG.ORA.CTL(LOAD2)\''
```

Notice that the filespec apostrophes in the example are doubled, which is required to get an apostrophe through the JCL PARM mechanism, as well as being escaped with a preceding backslash for Loader's parser.

Command line parameters for SQL*Loader can easily exceed the 100-character maximum of the JCL PARM field or TSO CALL command. You can use the "++" parameter file mechanism described in [Chapter 2](#) to supply Loader parameters in a data set. SQL*Loader also has a generic parameter file feature that is similar to the "++" mechanism but is available on all platforms (and also works in a POSIX shell, which is not true of "++"). If you use the generic parameter file, the filespec supplied for the PARFILE parameter is not subject to name extension before opening. This file can be a sequential or instream (DD *) data set, a PDS member, or an HFS file. The file must not contain record sequence numbers or other data that is not a SQL*Loader command line parameter.

SQL*Loader message output (including the aforementioned prompt and help message) is written to C standard output. Detailed information about load processing is discussed in the sections that follow.

SQL*Loader Files and Filespecs

SQL*Loader uses several types of files. In a single execution of Loader, multiple files of some types may be used. In some cases, if you do not supply a filespec for a file that is required, Loader derives a filespec for the file based on some other filespec that you did supply. What gets derived depends on the supplied filespec and on the environment in which Loader is running with POSIX OFF or ON.

The filespec derivation schemes mean care must be taken when running SQL*Loader with filespecs that are counter to the POSIX expectation: HFS files with POSIX OFF or data sets with POSIX ON. Deriving a data set filespec from an HFS filespec, or an HFS filespec from a data set filespec, is not supported. In these situations, filespecs must be supplied explicitly rather than being derived.

Control File

Exactly one SQL*Loader input control file, mentioned earlier, is required. The control file can be a sequential data set, PDS member, instream (DD *) data set, or HFS file. A data set must have a record format of F, FB, V, or VB. Records in a data set must not have sequence numbers or other data that is not part of Loader's control file syntax. There is no default filespec for the control file; if it is not supplied on the command line or PARM, a prompt is issued and the filespec is read from C standard input. The

filespec for the control file is subject to extension processing with the suffix `ctl`. As discussed in [Chapter 2](#), this may affect filespecs that specify a data set name or HFS file name; it does not affect a `//DD:` filespec.

Log File

An execution of Loader also requires one output log file, to which load statistics, informational, and error messages are written (separate from what is written to C standard output). The log filespec can be supplied on the command line or PARM and can specify a sequential data set, PDS member, SYSOUT, HFS file, or a `//DD:` filespec that resolves to any of these. If none is supplied, the log filespec is derived from the control filespec as follows:

- If the control filespec is a `//DD:` type, the log filespec is `//DD:LOG`.
- If the control filespec is a data set name type, the log file is written as a data set named `root.LOG`, where `root` is the control file data set name with any "directory" and extension suffix (as defined for data set names) removed. If the control filespec included a PDS member name in parentheses, the same member name specification is included at the end of the log filespec. This data set name acquires a high-level qualifier (TSO PROFILE PREFIX or, elsewhere, the z/OS user ID) when opened, so the ultimate data set name used is `hlq.root.LOG`.
- If the control filespec is an HFS file name, the log file is written as an HFS file named `root.log`, where `root` is the control file name with any directory path and extension suffix (as defined for HFS file names) removed. This file is written in the current working directory.

In the latter two cases, it is the file name extension mechanism that supplies the `.LOG` or `.log` suffix. This means FNA processing can be applied when the log file is a data set.

If your control filespec is counter to the POSIX expectation, the log filespec must not be derived—it must be supplied explicitly.

Data Files

An execution of SQL*Loader uses one or more input data files containing data to be loaded into Oracle database tables. One data filespec can be supplied on the command line or PARM, or in the first INFILE or INDDN clause in the Loader control file input. Additional data filespecs can be supplied using additional INFILE or INDDN clauses. If you do not supply any data filespec, a default is derived from the control filespec. The derivation is similar to that for log files, just described, but uses the suffix `.DAT` or `.dat`. When the control filespec is a `//DD:` type, the derived data filespec is `//DD:DATA`.

If your control filespec is counter to the POSIX expectation, the data filespec must not be derived—you must supply it explicitly on the command line or in the Loader control file.

On z/OS, a SQL*Loader data file can be a sequential data set, PDS member, instream (DD *) data set, VSAM entry-sequenced (ESDS), key-sequenced (KSDS), or relative record (RRDS) cluster, or an HFS file. Sequential or partitioned data sets must have a record format of F, FB, V, or VB. The special control file notation "INFILE *" is used to indicate that input data is in the Loader control file itself, following the control statements. In this situation, "*" is not interpreted as a filespec for the TSO terminal (as it normally would be in a TSO POSIX OFF environment). To specify a data file as the TSO terminal you must use `//*` or a DD statement allocated to the terminal.

When you supply a data filespec on the command line or PARM or using an INFILE clause, any of the available valid filespec types can be used, including //DD: or DD:.. If you supply a data file with the INDDN clause, the value you supply must be a valid 1-character to 8-character DD name without a //DD: or DD: prefix. Only DD names can be supplied using INDDN.

The control file allows you to supply something called a "file-processing options string" in the INFILE or INDDN clause. On z/OS, this string is used only with HFS files and takes the same options used on UNIX platforms, described in the *Oracle Database Utilities* manual. (It specifies any of several ways to parcel an HFS file into records, which is not a native HFS concept.) With z/OS data sets the processing options string is ignored. Records are a native concept with data sets, and what the generic SQL*Loader documentation calls a physical record maps to what z/OS calls a logical record in a data set.

Bad and Discard Files

For each data file that is processed, SQL*Loader may write a bad file and a discard file. The former contains copies of input records that contained data errors or were rejected by the Oracle server due to data-related errors (such as a unique key constraint violation). The latter file contains records that did not meet selection criteria specified in the WHEN clause in the control file.

Both files are used conditionally and are not opened unless circumstances require it. The bad file is opened if one or more input records from a data file contain data errors. The discard file is opened if one or more input records fail to meet WHEN clause criteria and you supplied a discard filespec or specified the DISCARDMAX option.

You can specify the bad file using BADFILE or BADDN and you can specify the discard file using DISCARDFILE or DISCARDN. When you use the "-FILE" keyword form, any of the supported filespec types can be supplied, including ones beginning with //DD: or DD:.. If you use the "-DN" keyword form, the value you supply must be a valid 1-character to 8-character DD name with no //DD: or DD: prefix. Only DD names can be specified using BADDN and DISCARDN.

When you supply filespecs for these files and Loader encounters input records that cause them to be opened, the filespecs are subject to extension processing. The bad file uses the extension `bad` and the discard file uses `dsc`.

When you do not supply filespecs for these files and the load needs to use them, Loader derives the filespecs based on the filespec of the associated data file. When the data filespec is derived from that of the control file, as described previously, this derivation is based on the derived data filespec. Bad and discard filespec derivation works as follows:

- For the first (or only) data file in a load, if the data filespec is a //DD: type, the derived bad filespec is //DD:BAD and the derived discard filespec is //DD:DISCARD. When a load involves multiple data files (with multiple INFILE or INDDN clauses), if the second or subsequent data filespec is a //DD: type, the derived bad filespec is //DD:BADn and the derived discard filespec is //DD:DISCARDn where n is a 1-digit or 2-digit decimal relative data file number within the control file ("2" for the second data file specified, "3" for the third, "11" for the eleventh, and so on). At most 99 bad and discard files are supported by this scheme. All data files in a load (including non-DD types) are counted for purposes of determining the relative data file numbers.
- If a data filespec is a data set name type, the bad filespec is derived as a data set named `ctlhlq.root.BAD` and the discard filespec is derived as `ctlhlq.root.DSC`, where `ctlhlq` is the "directory" (high-level qualifier) from the control filespec and

root is the data file data set name with any "directory" and extension suffix removed. If the data filespec included a PDS member name in parentheses, the same member name specification is included at the end of the bad and/or discard filespec.

Use of the control filespec "directory" in these derivations mirrors what is in the control filespec: if the filespec contains a quoted data set name with an explicit high level qualifier, the same qualifier (and quotes) are used in the derived filespecs. If the control filespec is not quoted and subject to PROFILE PREFIX or user ID prefixing, the same is true of the derived filespecs.

- If a data filespec is an HFS file name, the bad file is derived as an HFS file named root.bad and the discard filespec is derived as root.dsc, where root is the data file name with any directory path and extension suffix (as defined for HFS file names) removed. This file is written in the directory associated with the control file, not the directory of the data file.

If either your control or data filespec is counter to the POSIX expectation, the bad and discard filespecs must not be derived—they must be supplied explicitly on the command line or in the Loader control file.

VSAM Data File Processing Considerations

The following considerations pertain to loads where a data file is one of the supported VSAM cluster types (KSDS, ESDS or RRDS).

- Input records are read in normal forward sequence for the cluster type: key sequence for KSDS, RBA sequence for ESDS, and record number sequence for RRDS. Empty slots in an RRDS cluster are not read and do not contribute to Loader SKIP or LOAD counts.
- While you can specify an AIX PATH for a load, causing records to be read in alternate key sequence, this impacts load performance and is not recommended.
- Loading from a VSAM object that is also open for update in another z/OS job or address space (including CICS TS and IMS TM address spaces) is not supported. Record Level Sharing (RLS) is not supported.
- Loading from VSAM objects containing invalid Control Intervals (lacking valid CIDF/RDF fields) is not supported.
- SQL*Loader provides no mechanism for specifying VSAM cluster passwords.

Bad and Discard File Attributes

The bad and discard files, when written, contain subsets of the data from the associated data file. On z/OS, these files must be the same major type (data set or HFS file) as the associated data file. When using HFS files, the bad and discard files are written with the same line terminator and related attributes that you specified (or defaulted) in the file-processing options string for the data file.

When you load data from a data set, you can allow the DCB attributes of the bad and discard files to default or you can override them by coding them explicitly on a DD statement or by using an existing data set with established DCB attributes (which Loader will preserve). When allowed to default, the RECFM and LRECL will match those of the associated data file. If you override these attributes you must ensure that the output record length is sufficient. For example, if the input data file has fixed record format (F or FB) and LRECL=100 and you want to use RECFM=VB for the bad file, you must use an LRECL of at least 104 for the bad file (100 bytes for data and 4 bytes for the record descriptor word).

When an input data file is a VSAM data set, the bad and discard files will be nonVSAM sequential data sets. (Loader cannot create a VSAM bad or discard file.) As with nonVSAM data set input, the bad and discard file DCB attributes will default to values that are appropriate for the VSAM cluster and its maximum RECORDSIZE attribute. If you override these attributes you must ensure that the LRECL is adequate.

If you override bad or discard DCB attributes with a fixed (F or FB) format and a data file record to be written is shorter than the fixed record length, the record is padded with binary zeroes to the required length.

SQL*Loader Return Codes

Refer to the *Oracle Database Utilities* manual for information about conditions in which return codes are set. The keyword return codes listed there have the following values on z/OS:

Keyword Return Code	Description	Value
EX_SUCC	success	0
EX_WARN	warning	4
EX_FAIL	fatal error	8

SQL*Loader Examples

In the following examples, [Example 1](#) is a z/OS UNIX shell execution of SQL*Loader, loading data from an HFS file into an Oracle instance on the same z/OS system. It assumes the environment variables ORACLE_HOME, PATH, and LIBPATH are already set appropriately. ORACLE_SID is set to enable cross-memory (XM) connection to the instance whose SID is TDB3.

Example 1

```
$ export ORACLE_SID=TDB3 [Text entered by user]
$ ls -l mydata.* [Text entered by user]
-rw-r--r-- 1 RJONES PGACCT 181203 Nov 5 15:52 mydata.dat
$ cat myload.ctl [Text entered by user]
LOAD DATA
  INFILE 'mydata'
  DISCARDMAX 10
  INTO TABLE SITE_REVIEW
  WHEN DATA3 != '500'
  FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
  TRAILING NULLCOLS
  (DATA1,
   DATA2 CHAR "NVL(:C, '****')",
   DATA3)
$ sqlldr control=myload userid=bsmyth/pinot [Text entered by user]
SQL*Loader: Release 10.2.0.5.0 - Production on Fri Feb 3 01:17:27 2012
Copyright (c) 1982, 2010, Oracle. All rights reserved.
Commit point reached - logical record count 1
Commit point reached - logical record count 2
Commit point reached - logical record count 3
Commit point reached - logical record count 6
Commit point reached - logical record count 7
$ ls -l mydata.* *.log [Text entered by user]
-rw-r--r-- 1 RJONES PGACCT 181203 Nov 5 15:52 mydata.dat
-rw-r--r-- 1 RJONES PGACCT 594 Mar 17 10:26 mydata.dsc
-rw-r--r-- 1 RJONES PGACCT 6288 Mar 17 10:26 myload.log
```

\$

Example 2 is a batch jobstep execution of SQL*Loader. It is similar to the previous example except for the environment. All filespecs except the control file have been allowed to default.

Example 2

```
//LOAD      EXEC PGM=SQLLDR,REGION=0M,
//      PARM='userid=bsmyth/pinot control=DD:ctl'
//STEPLIB DD DISP=SHR,DSN=ORACLE.V10G.COMDLOAD
//ORA$LIB DD DISP=SHR,DSN=ORACLE.V10G.MESG
//ORA@TDB3 DD DUMMY
//CTL      DD *
LOAD DATA
  DISCARDMAX 10
  INTO TABLE SITE_REVIEW
  WHEN DATA3 != '500'
  FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
  TRAILING NULLCOLS
  (DATA1,
   DATA2 CHAR "NVL(:C, '****')",
   DATA3)
/*
//DATA      DD DISP=SHR,DSN=GKSC.MASTER.SPR
//LOG       DD SYSOUT=*
//DISCARD   DD DISP=(,CATLG),DSN=BSMYTH.LOAD.DSC,UNIT=SYSDA,
//      SPACE=(TRK,(10,10),RLSE)
```

Export and Import

Export and Import are complementary Oracle utilities used chiefly to transport Oracle database data between Oracle databases and between systems. Export writes a sequential file that is a "transportable copy" of database tables, indexes, and other objects along with their metadata descriptions, such as table and column names and data types. Import reads the sequential file produced by export, defining the database objects in the target database and then loading the data (rows, index entries, or other contents).

Datapump Export and Import utilities, described in the section ["Datapump Export and Import"](#) on page 4-17 provide functions similar to Export and Import with additional features and capabilities.

For more information about Export and Import and Datapump Export and Import, refer to *Oracle Database Utilities*.

Invocation

On z/OS, Export and Import support batch job, TSO, and shell execution environments. JCL procedures for batch execution (default names ORAEXP and ORAIMP) are supplied by Oracle and may be installed on your system with the same or different names. In TSO, both CALL and command processor (CP) invocation are supported. For the non-POSIX environments, the load module or program object names are EXP and IMP. In a z/OS UNIX shell, use the `exp` and `imp` (lowercase) commands to invoke these utilities.

Parameters for Export and Import can be supplied on the command line or PARM field or in a file. If parameters are supplied in neither place, these utilities prompt for

required inputs; prompts are written to C standard output and the responses are read from standard input.

The parser used by Export and Import for parameters has its own syntax conventions (described in the generic documentation) and is sensitive to apostrophes (single quotes). If you use apostrophes in a filespec parameter, you must "escape" them with a preceding backslash to keep the parser from trying to interpret them as syntax elements. For example, to specify the FILE= parameter with the data set name filespec `//'JSMITH.ORADB1.DMP'` in a batch Export job, use the following code:

```
//EXPORT EXEC PGM=EXP,
//  PARM='scott/tiger file=//\'JSMITH.ORADB1.DMP\''
```

Notice that the filespec apostrophes in the example are doubled, which is required to get an apostrophe through the JCL PARM mechanism, as well as being escaped with a preceding backslash for Export's parser.

If command line parameters for Export and Import exceed the 100-character maximum of the JCL PARM field or TSO CALL command, you can use the "++" parameter file mechanism described in the section ["Parameters in Batch \(JCL\)"](#) on page 2-6 to supply parameters in a data set. Export and Import also have a generic parameter file feature that is similar to the "++" mechanism but is available on all platforms (and also works in z/OS UNIX shell, which is not true of "++"). If you use the generic parameter file, the filespec supplied for the PARFILE parameter is not subject to name extension before opening. This file can be a sequential or instream (DD *) data set, a PDS member, or an HFS file. The file must not contain record sequence numbers or other data not part of Export or Import's command line parameters.

For certain operations, Export and Import require a special user ID/password parameter that contains spaces, such as `/ AS SYSDBA`. Refer to the section ["Parameters Containing Spaces"](#) on page 2-9 for information about how to specify such values.

Export File

The file written by Export and read by Import is called an export file or dump file. Filespecs that you supply for this purpose are subject to extension processing with the suffix `dmp`. If you don't supply a filespec (you omit the FILE parameter and/or supply an empty line at the file prompt) the default `expdat.dmp` is used in both POSIX OFF and POSIX ON environments. In a POSIX OFF environment this is processed as a data set named `EXPDAT.DMP` (with prefixing implied). In a shell, it is treated as an HFS file named `expdat.dmp` in the current working directory.

When the export file is a data set, DCB attributes must be established when the Export utility opens the file. If you do not specify DCB attributes on a DD statement or TSO ALLOC command, and you do not use an existing data set with established attributes (which Export will preserve), LE default attributes are used. Unlike most Oracle tool and utility files, the export file is opened in "binary" mode and is subject to different default attributes than those described in the section ["Data Set DCB Attributes"](#) on page 2-27. For both disk and tape devices, the default attributes for output binary files are `RECFM=FB` and `LRECL=80`.

If you override the DCB attribute defaults by supplying them on a DD or ALLOC command or by using an existing data set, you can use either fixed- or variable-length record formats and any LRECL allowed by LE. However, if you are creating an export file that will be processed by Import running on a non-z/OS system, you may want to avoid variable record formats (V or VB) depending on how the data will be moved to the target system. Either the transporting software (for example, FTP) or the Import

utility on the target system may have difficulty with the imbedded record and block descriptors used by V and VB formats. Refer to the section "[Cross-System Export/Import](#)" on page 4-16 for additional details.

The RECORDLENGTH parameter of Export and Import does not affect or relate to the DCB attributes of the export file.

Cross-System Export/Import

One of the strengths of Export and Import is that they can be used to move Oracle data between dissimilar platforms without an Oracle Net connection between the two. This may be faster, and in some cases more secure, than running one utility or the other over an Oracle Net connection.

When Import reads data that was created by Export running on a different platform, the data must be unmodified from what was written. (Translation of data between formats and character sets is handled automatically by Import.) If you use something like File Transfer Protocol (FTP) software to move the data to the target system, specify a "binary mode" or similar processing option to prevent attempts to translate character data. This requirement exists when moving non-z/OS Export data to z/OS and when moving z/OS Export data to a non-z/OS system. If you fail to do this and the data is translated, Import typically issues the following message:

```
IMP-00010: not a valid export file, header failed verification
```

Export and Import Return Codes

Refer to *Oracle Database Utilities* for information about conditions in which return codes are set. The keyword return codes listed there have the following values on z/OS:

Keyword Return Code	Description	Value
EX_SUCC	success	0
EX_WARN	warning	4
EX_FAIL	fatal error	8

Unsupported Features

Cross-platform Export/Import of a partitioned table between EBCDIC (z/OS) and ASCII systems will fail if the partitioning is based on character ranges that are incompatible between EBCDIC and ASCII. If your partition ranges are not compatible and retain the same sequence between ASCII and EBCDIC, you must manually create the table with different partition criteria (or with no partitioning) in the target database and then run Import with the IGNORE=Y option so the error on Import's table create is ignored.

Export and Import Examples

The following example shows a batch jobstep execution of Export. This is a simple export of all objects owned by a user ID:

```
//EXP EXEC PGM=EXP,
//  PARM='hrmdba/satchel file=dd:expout'
//STEPLIB DD DISP=SHR,DSN=ORACLE.V10G.CMDLOAD
//ORA$LIB DD DISP=SHR,DSN=ORACLE.V10G.MESG
//ORA@RM5 DD DUMMY
```



```
//EXPOUT DD DISP=(,CATLG),DSN=BMANRY.EXPTEST2.DMP,
// UNIT=SYSDA,SPACE=(TRK,(10,10),RLSE),
// DCB=(RECFM=FB,LRECL=120)
```

Datapump Export and Import

Datapump Export and Import are functionally similar to Export and Import discussed previously, but all of the I/O processing for dump files is done in the Oracle database server rather than in the client utility session.

Datapump Export and Import provide capabilities that Export and Import do not, in particular an easy way to spread processing over multiple tasks to improve parallelism. For more information about Datapump Export and Import, refer to *Oracle Database Utilities*.

Besides using the Datapump client utilities, you can invoke Datapump Export and Import programmatically using the DBMS_DATAPUMP PL/SQL package. For more information about the DBMS_DATAPUMP package, refer to the *PL/SQL Packages and Types Reference*.

On z/OS, whether invoked using a utility or the DBMS_DATAPUMP package, Datapump Export and Import can only process files in the HFS. This includes the dump files written by Datapump Export or read by Datapump Import and the text log file written by both components.

Security for Datapump file operations is provided using database directory objects, which must be created and granted to users by a database administrator before Datapump Export and Import can be used. If you are unable or choose not to use the HFS file system for exporting or importing or if your database administrator has not created directory objects, you can use the Export and Import utilities described in the previous section.

Invocation

On z/OS, Datapump Export and Import support batch job, TSO, and shell execution environments. JCL procedures for batch execution (default names ORAEXD and ORAIMD) are supplied by Oracle and may be installed on your system with the same or different names. In TSO, both CALL and command processor (CP) invocation are supported. For the non-POSIX environments, the load module or program object names are EXPDP and IMPDP. In a z/OS UNIX shell, use the `expdp` and `impdp` (lowercase) commands to invoke these utilities.

Parameters for Datapump Export and Import can be supplied on the command line or PARM field or in a file. If parameters are supplied in neither place, these utilities may prompt for certain inputs. Prompts are written to C standard output and the responses are read from standard input.

If command line parameters for Datapump Export and Import exceed the 100-character maximum of the JCL PARM field or TSO CALL command, you can use the "++" parameter file mechanism described in the section "[Parameters in Batch \(JCL\)](#)" on page 2-6 to supply parameters in a data set. Datapump Export and Import also have a generic parameter file feature that is similar to the "++" mechanism but is available on all platforms (and also works in a shell, which is not true of "++"). If you use the generic parameter file, the filespec supplied for the PARFILE parameter is not subject to name extension before opening. This file can be a sequential or instream (DD *) data set, a PDS member, or an HFS file. The file must not contain record sequence numbers or other data not part of Datapump Export or Import's command line parameters.

For certain operations, Datapump Export and Import require a special user ID/password parameter that contains spaces, such as "/ AS SYSDBA". Refer to the section ["Parameters Containing Spaces"](#) on page 2-9 for information about how to specify such values.

Datapump Export and Log Files

The file that contains the database data written by Datapump Export and read by Datapump Import is called an export file or dump file. While conceptually similar to the export files of the Export and Import utilities described previously, the two types are not interchangeable. For example, you cannot read a plain Export file with Datapump Import and you cannot read a Datapump Export file with plain Import.

Besides writing or reading an export file, the execution of Datapump Export or Import normally writes a log file. This is just a text log reporting processing details.

On z/OS, the export and log files used by Datapump Export and Import must be HFS files; they cannot be z/OS data sets.

Because Datapump export and log file processing is done in the Oracle database server rather than the client job or session, access to both types of file is controlled with database directory objects, which are created by a database administrator. Although there are forms of directory object for both HFS files and data sets, as discussed in [Chapter 5, "Oracle Server Considerations on z/OS,"](#) only the HFS form of directory object can be used in a Datapump operation.

The user-supplied names for the export and log files are required to begin with "./" (dot-slash) so that they are seen as unambiguous HFS filespecs. This requirement is unique to z/OS Oracle database servers and is not accepted by Oracle database servers on other platforms. (Oracle normally prohibits any kind of path prefix on user-supplied filenames that are processed in the database server.)

When supplied by the user, the export and log file names are subject to extension processing with the suffixes `.dmp` and `.log`, respectively. Because these are always treated as HFS files, extension processing is under HFS rules, as described in [Chapter 2, "Oracle Software Interaction with z/OS and Language Environment,"](#) and FNA processing is not available. If omitted, the default name for the export file is `./expdat.dmp` and for the log file it is `./export.log` (Datapump Export) or `./import.log` (Datapump Import). These files are always accessed in an HFS directory from a database directory object. The directory object name is supplied with the file name (separated by a colon) or separately, in the DIRECTORY command line parameter.

The following example shows Datapump Export being invoked in TSO to dump tables ACCT1_MST and ACCT1_DET for user RACHELQ:

```
READY
expdp rachelq/qow00a tables=(acct1_mst,acct1_det) directory=acctdir -
dumpfile=./acctexp
```

```
Export: Release 10.2.0.5.0 - Production on Monday,...
```

Both the export file and the log are written in the HFS directory associated with the ACCTDIR directory object, to which RACHELQ was previously granted access by a database administrator. The export file has root name `acctexp`. After extension processing the name is `acctexp.dmp`. The log file defaults to `./export.log`.

Datapump Processing in the Server

When you use Datapump, much of the server processing is performed by auxiliary processes (z/OS subtasks) in the server address space. The number of processes involved depends on the degree of parallelism requested or permitted by the database administrator. Processing by auxiliary subtasks runs in the server's WLM goal and is accounted to the server address space, not to the client.

Datapump Export and Import Interactive Mode

Datapump Export and Import utilities have an interactive mode that is initiated by interrupting the utility after server processing begins. In TSO, this is accomplished with an ATTN or PA1 signal. In a shell, use Ctl-C or a similar mechanism to interrupt the utility. Interactive mode is not supported in z/OS batch environments.

Datapump Export and Import Return Codes

Datapump Export and Import can end with normal success, success with one or more warning conditions, or be terminated by an error. The return codes associated with these conditions are as follows:

Return Code	Value
Success	0
Warning	4
Fatal Error	8

In the event of a warning or error return code, check the Datapump log file for messages describing the warning or error condition.

TKPROF

TKPROF is a utility used in tuning Oracle SQL statements. It reads trace files produced by the Oracle database server and formats the trace data into a report on the execution path and processing statistics associated with SQL statements from an Oracle database session. Optionally, it can connect to a target server and issue EXPLAIN PLAN requests for the subject SQL statements, and it can write a separate file containing SQL statements used by an application. For more information about TKPROF, refer to the *Oracle Database Performance Tuning Guide*.

Invocation

TKPROF on z/OS supports batch job, TSO, and shell execution environments. No batch JCL procedure is provided, but one can be created easily, if necessary. In TSO, both CALL and command processor (CP) invocation are supported. For non-POSIX environments, the program object name is TKPROF. In a z/OS UNIX shell, use the `tkprof` command (lowercase) to invoke this tool.

TKPROF requires two command line parameters, the first identifying the input Oracle trace file to be read and the second supplying a filespec for the output report. If no parameters are supplied, TKPROF displays a brief help message and ends normally. If the input trace file is specified but not the output file, TKPROF prompts (to standard output) for the output filespec and reads it from standard input. Additional command line parameters are permitted for various options.

Input Trace File

Input to TKPROF is a flat file written by the Oracle database server when SQL tracing is active in a client session. For information about methods for activating and deactivating the trace with an ALTER SESSION statement or the DBMS_SESSION.SET_SQL_TRACE procedure, refer to the *Oracle Database Performance Tuning Guide*.

On z/OS, Oracle database server trace files are written in one of two forms: as SYSOUT (JES) spool files or as sequential (DSORG=PS) disk data sets. TKPROF cannot read JES spool files directly, so if your installation is using SYSOUT for trace files, you will need to identify the trace file containing the SQL trace and copy it to a sequential disk data set to use TKPROF. Copying can be done with a tool such as the PRINT command of the IBM Spool Display and Search Facility (SDSF) or a comparable mechanism. Whether the trace files for your database instance are written to SYSOUT or directly to a data set, you may need to work with your database administrator to obtain access to the trace data.

- It is a limitation of IBM access methods that you cannot run TKPROF (or any other program) to directly access an Oracle database server trace that is being written as a disk data set while the database server still has the data set open for output. To ensure that a trace file is closed, the database server session associated with the trace must end (disconnect from the database server). This is one of the advantages in using spool files for database server tracing; tools such as IBM SDSF can read and copy a spool file while it is still open to the database server.

Note: The *Oracle Database Performance Tuning Guide* discusses several Oracle database server initialization parameters that are relevant to SQL tracing and TKPROF. Two of these, MAX_DUMP_FILE_SIZE and USER_DUMP_DEST, are not meaningful on z/OS and can be ignored. The third, TIMED_STATISTICS, is meaningful and may need to be set to get useful performance data. TIMED_STATISTICS is a dynamic parameter and can be set at the session level.

Although the input trace file is normally going to be a sequential data set on z/OS, TKPROF will also read a PDS member or an HFS file as its input. File name extension processing with the suffix `trc` is done on the trace filespec before it is opened. If your trace data set name does not end in `.TRC`, include two slashes in front of the name or use a `DD: filespec` to avoid extension processing.

Output Files

The report file written by TKPROF can be a sequential disk data set, a PDS member, a SYSOUT (spool) data set, or a file in the HFS. The filespec you supply for the output file is subject to extension processing with the suffix `prf`. By default, records in this report are at most 80 data bytes long; if you use the WIDTH command line option to request report lines that will exceed the LE default LRECL (1028, or 1024 data bytes), you must override the file's LRECL using DD statement parameters or FNA.

Optionally, using the RECORD= parameter, TKPROF will write a file containing all user SQL statements from the session. Like the report, this file can be a sequential disk data set or PDS member, a SYSOUT data set, or a file in the HFS. Filespecs supplied using the RECORD parameter are subject to extension processing with the suffix `sql`.

Return Codes

TKPROF either processes successfully or encounters errors and fails, there are no warning conditions. Return codes from TKPROF are as follows:

Return Code	Value
Success	0
Failure	8

TKPROF Example

The following example shows a batch job step to run TKPROF to analyze a trace file in a z/OS data set named JSMITH.ORAPDB1.SESS3.TRACE:

```
//TKPROF EXEC PGM=TKPROF,REGION=8192K,  
// PARM='DD:TRACE S:C'  
//STEPLIB DD DISP=SHR,DSN=ORACLE.V10.COMDLOAD  
//ORA$LIB DD DISP=SHR,DSN=ORACLE.V10.MESG  
//TRACE DD DISP=SHR,DSN=JSMITH.ORAPDB1.SESS3.TRACE
```

The report is written directly to SYSOUT class C using a SYSOUT filespec.

Oracle Server Considerations on z/OS

If you are writing SQL, PL/SQL, or Oracle server Java that will run on a z/OS Oracle database server, most of what you do will be exactly the same as for any other Oracle platform. The few distinctions that exist are in areas that should be no surprise: character data (EBCDIC versus ASCII) is one, and access to operating system files (or z/OS data sets) is another. Here we explain aspects of these areas that might influence your design or coding.

This chapter contains the following sections:

- [Character Data](#)
- [File Processing in the Server](#)

Character Data

An Oracle database has an associated character set, an attribute that is specified (or defaulted) when the database is first created, on the CREATE DATABASE statement. Application data in character-type table columns (such as CHAR and VARCHAR.) and database object identifiers (for example, table and column names) are stored using this character set. On a given Oracle platform, all supported database character sets are from the same family: either ASCII or EBCDIC. On z/OS, of course, the EBCDIC family is used. The default Oracle database character set on z/OS is what IBM calls EBCDIC codepage 1047, which Oracle identifies as WE8EBCDIC1047. Other popular EBCDIC codepages are available and can be used on z/OS. Refer to the *Oracle Database Globalization Support Guide* for additional information about database character sets.

Character Data and SQL Queries

Differences between EBCDIC and ASCII become noticeable in certain types of SQL query. The most obvious is the use of ORDER BY or GROUP BY on character values in a SELECT to return results in a particular order. EBCDIC and ASCII have different sort order for various groups of characters. For example, decimal digits (0-9) sort before all letters in ASCII, but after all letters in EBCDIC. Upper case letters sort before lowercase in ASCII, but after lowercase in EBCDIC. There are other differences in the sorting of punctuation and other symbols.

The consequence of these differences is that, given identical table data, a SELECT with an ORDER BY or GROUP BY clause involving character data can return results from a z/OS Oracle server in a different order than an ASCII-based Oracle server. In most cases this is not a problem and in fact is exactly what you want. When it is not, you can use the SQL function NLSSORT or the Oracle server parameter NLS_SORT (which can be set at the session level) to specify a different collating sequence for ORDER BY or

GROUP BY processing. Both NLSSORT and NLS_SORT are discussed in more detail in the *Oracle Database Globalization Support Guide*.

A more subtle difference—one that sometimes affects applications that are migrated from an ASCII server to EBCDIC or vice versa—concerns the use of ordinal comparisons on character data in the WHERE clause of a SQL statement. Consider this SELECT statement:

```
SELECT ACCT_NUM, BALANCE FROM LEDGER_ACTIVE WHERE CATEGORY > 'BR10'  
       AND CATEGORY < '99AX'
```

We'll assume that CATEGORY is a CHAR(4) column containing an encoded value comprising both digits and letters. This SELECT might work fine with an EBCDIC server but it will never return any rows from an ASCII server because in ASCII, digits are numerically lower than letters. (There is no ASCII character string that is greater than 'BR10' and less than '99AX'.)

Similar issues surround the use of ordinal operators and character data in PL/SQL and Java stored procedures. If you are writing applications that will be migrated between EBCDIC and ASCII Oracle servers, you must avoid this kind of construct or be prepared to change the application when migrating.

Character Data and Partitioned Tables

Partitioned tables are divided into multiple physically-separated groups of rows according to data criteria. If you partition a table by range of values in a character table column, you are establishing an ordinal comparison on rows that are inserted into the table. As with the SELECT...WHERE clause described in the prior section, it is possible to define partition ranges that work in EBCDIC but not ASCII, or vice versa. Care must be taken when designing a range-partitioned table that you expect to move between a z/OS Oracle server and one on an ASCII platform. Refer to the *Oracle Database Administrator's Guide* for details on creating and using partitioned tables. Refer to the discussion of limitations on the Oracle Export/Import utilities when processing range partitioned tables, in [Chapter 4, "Oracle Tools and Utilities on z/OS."](#)

Characters in SQL Statements

Most of the characters used as syntax elements in SQL translate readily between ASCII and EBCDIC, making it easy to move SQL between z/OS and other platforms. One character that sometimes causes trouble is the vertical bar, used in the SQL concatenation operator. There are two different vertical bars in some EBCDIC codepages, including the default Oracle code page on z/OS (1047). Oracle for z/OS accepts the solid vertical bar (hexadecimal 4F in most codepages) as the SQL concatenation operator; the split vertical bar (hexadecimal 6A) is not accepted.

Some file transport mechanisms (for example, some versions of FTP) may translate an ASCII vertical bar to codepoint 6A when moving a file to an EBCDIC system. There may also be terminal emulators that produce codepoint 6A in a z/OS session when the vertical bar key is pressed. This can result in invalid SQL statements for a z/OS Oracle server. One way to avoid this is to use the keyword function CONCAT in SQL statements instead of the vertical bar. CONCAT provides exactly the same functionality as vertical bar concatenation. For additional information about CONCAT refer to the *Oracle Database SQL Reference*.

File Processing in the Server

Before reading this, you should be familiar with the file name and file processing concepts and features discussed in [Chapter 2](#).

Much of the file processing in the Oracle server concerns the files in which the Oracle database is stored. These files, which are VSAM Linear (LDS) clusters on z/OS, are a database administration concern and are not normally referenced directly by applications. Considerations for these files are covered in the *Oracle Database System Administration Guide for IBM z/OS on System z*.

There are, however, file access mechanisms in the Oracle server that are used directly by applications. The PL/SQL UTL_FILE package, Java file I/O, external LOBs, external tables, and Datapump Export and Import are capable of accessing more or less arbitrary files external to the Oracle database. When an application uses these features, file accesses are attempted on the application's behalf by the Oracle server address space. Here we cover considerations for these features that are specific to z/OS.

Relative to file processing concepts described in [Chapter 2](#), two key points govern file processing behavior in the server:

- The POSIX indicator in a z/OS client application, tool, or utility does not affect processing in the Oracle server. How filespecs are interpreted in the server (particularly ambiguous filespecs, as defined in [Chapter 2](#)) depends on the particular server feature, not on whether the client is running in TSO/batch or a shell. Some server features are limited to accessing HFS files only or data sets only, while others are capable of accessing both, distinguished by filespec syntax.
- No default prefixing or directory path applies to file names in the server. There is no implicit z/OS user ID prefixing of data set names, and there is no concept of a current working directory for HFS file references.

Security Considerations

Both Oracle security and native z/OS security are involved when Oracle accesses a file on behalf of your application. Oracle security typically involves specific database objects and mechanisms: for example, in order to access a file using the external table feature, the Oracle user ID involved must be granted appropriate access rights on the database directory object associated with the file. This check occurs entirely within the Oracle database server.

When the server actually opens the file, z/OS security checks occur. These checks are based on the z/OS user ID associated with the server address space, not with an Oracle user ID nor with the z/OS user ID of your application job or session (if it is running on z/OS). One z/OS user ID is associated with an Oracle server running on z/OS; this usually is set up by your system or security administrator.

The exact nature of the native z/OS security check depends on the kind of file being opened and on how the z/OS security component (for example, RACF or a comparable product) is configured. Opening a data set performs the normal data set name-based check for the requested mode of access (generally either read or write). Opening an HFS file performs a check to enforce the permission settings associated with the HFS directory and the file.

In short, both types of security check—Oracle and native z/OS—must succeed in order for an application's server file access operation to proceed. If the Oracle check fails, a specific Oracle error for that condition is issued. If the native z/OS check fails, it generally is reported as an error in a "file open" function with varying amounts of

accompanying detail. z/OS error reporting details are discussed in each of the sections that follow.

Database Directory Objects

Several of the file access features discussed in this section depend on the database directory object as a security and control mechanism. Accessing a file with these features requires both a directory object identifier and a file name; the directory object must exist in the database and the Oracle user ID must have been granted appropriate access rights on the directory object. Creating directory objects and granting access to them is usually a database administrator or security administrator responsibility.

Associated with each directory object is a character string that relates to the OS file system. In a UNIX Oracle server, this string identifies a hierarchical file system (HFS) path. In z/OS, a directory object can identify either an HFS path or the leading (leftmost) portion of a data set name filespec. The former is used when accessing HFS files and the latter when accessing a data set. Not all Oracle features that use directory objects are capable of accessing both HFS files and data sets; refer to the following sections for specific feature details.

Because of the dual interpretation of directory objects on z/OS, the character string associated with the object must be unambiguous as to type, as that term is defined in [Chapter 2](#). If it represents an HFS path, it must begin with a single slash followed by zero or more HFS directory names, each ending with a slash. (The final ending slash is not required.) If the Directory represents high level qualifiers of a data set name, it must begin with exactly two forward slashes followed by one or more qualifiers, each ending with a period. The final ending period is not required; the string is assumed to contain whole data set name qualifiers, and an intervening period is added if needed when the string is used to compose a complete filespec. Including a leading single quote (apostrophe) between the slashes and the high level qualifier, signifying no user ID prefixing, is permitted but not required since no prefixing is done in the server.

A Directory defined with HFS syntax can be used only to access files in the z/OS POSIX HFS environment. One defined with data set syntax can be used only to access data sets.

The following example shows CREATE DIRECTORY statements for HFS file and data set access in turn:

```
CREATE DIRECTORY APP1DIR AS '/oracle/dev/app1';  
CREATE DIRECTORY APP2DIR AS '//JSMITH.APP2';
```

UTL_FILE

The UTL_FILE PL/SQL package allows you to read and write arbitrary flat files containing text data. On z/OS, UTL_FILE can access both HFS files and data sets, with data set access subject to limitations described in this section.

Prior to Oracle 10G, UTL_FILE relied on an Oracle server initialization parameter, UTL_FILE_DIR, to indicate which HFS directories could be accessed by the package, and data set access was not supported on z/OS. Starting in Oracle 10G, UTL_FILE uses the database directory object as the control mechanism. The UTL_FILE_DIR parameter is still supported, but Oracle recommends using directory objects instead. In order to access a data set with UTL_FILE, you must use a directory object defined as described in the previous section.

UTL_FILE data set support is limited to sequential (DSORG=PS) and partitioned (DSORG=PO) organizations with record formats F, FB, V, and VB. Partitioned data set members can be accessed for input only: opening a PDS member for output is not

supported. A given data set can be opened for input by multiple user sessions across multiple Oracle servers, or be opened for output by a single user session in one server. An open for output must specify a new (non-existent) data set; opening an existing data set for output is not supported.

When you access a data set with UTL_FILE, the filename argument passed to the FOPEN function is the trailing (right-hand) portion of a data set name. When opening for input, this can include a PDS member name in parentheses. Both the data set name and member name are case-insensitive. No other filespec syntax elements (such as leading slashes or surrounding apostrophes) are permitted in the filename. The filename is appended to the directory object string (with an intervening period and closing apostrophe if necessary) to form the complete filespec to be opened. Only data set name filespecs can be formed with this mechanism: SYSOUT and DD filespecs are not supported by UTL_FILE.

When accessing data sets, only the following UTL_FILE package functions are supported:

- FCLOSE
- FCLOSE_ALL
- FFLUSH (This is accepted for data sets, but has no effect. Buffer flush occurs as needed and when the file is closed.)
- FOPEN (Only modes "r" and "w" are accepted.)
- GET_LINE
- IS_OPEN
- NEW_LINE
- PUT
- PUTF
- PUT_LINE

UTL_FILE Example

In the following example, a SQL*Plus batch job is used to create a data set directory object, grant read access to user SCOTT, and then has SCOTT use the directory object to access a member of a PDS from an anonymous PL/SQL script. The PDS member that is accessed is JSMITH.TEST.CNTL(DATA1):

```
//PLUS      EXEC PGM=SQLPLUS,REGION=0M,PARM='/nolog'
//STEPLIB DD  DISP=SHR,DSN=ORACLE.V10.CMDLOAD
//ORA$LIB DD  DISP=SHR,DSN=ORACLE.V10.MESG
//ORA@ORA1 DD  DUMMY
//SYSIN     DD  *
whenever oserror exit 8
whenever sqlerror exit 8
connect dbauser/chuckle
create directory testdir as '//JSMITH.';
grant read on directory testdir to scott;
connect scott/tiger
set serveroutput on size 1000000
declare
    fd utl_file.file_type;
    buf varchar2(1024);
begin
    fd := utl_file.fopen('TESTDIR', 'test.cntl(data1)', 'r');
    begin
```

```
        loop
            utl_file.get_line(fd, buf, 1024);
            dbms_output.put_line(buf);
        end loop;
    exception
        when no_data_found then
            dbms_output.put_line('---EOF reached.');
```

```
        when others then
            dbms_output.put_line('some other read error');
```

```
    end;
    utl_file.fclose(fd);
exception
    when others then
        dbms_output.put_line('some fopen error');
```

```
end;
/
exit
/*
//
```

JAVA File I/O

Java provides various mechanisms for accessing arbitrary files, such as the `java.io` package. When you use Oracle Database Java, your Java procedures are processed on the server. On z/OS, all file accesses from Oracle Database Java procedures are treated as HFS file accesses. Refer to the *Oracle Database Java Developer's Guide* for more information about using Oracle Database Java.

External LOBs (BFILEs)

External LOBs are database large objects whose data resides in an external (non-database) file on the server system. Applications can access LOB data (both internal and external) using the PL/SQL `DBMS_LOB` package, OCI calls, and certain constructs in precompiler programs. Access to an external (file) LOB uses a `BFILE` type. The key identifying components of a `BFILE` are a database directory object name and the external file name, both supplied using the `BFILENAME` function. Refer to the *Oracle Database Application Developer's Guide - Large Objects* for more information about external and internal LOBs and `BFILE` processing.

On z/OS, files accessed as external LOBs must be HFS files. Although directory objects on z/OS are capable of designating a data set name prefix, only directory objects that specify an HFS directory can be used to identify a `BFILE`.

External Tables

External tables are Oracle database tables that have their row data in an external (non-database) location. Oracle provides two different access drivers to process external tables: the `ORACLE_LOADER` access driver and the `ORACLE_DATAPUMP` access driver.

The `ORACLE_LOADER` access driver functions similarly to `SQL*Loader`, enabling you to load normal Oracle database tables with data sourced from an external file. The `ORACLE_DATAPUMP` access driver facilitates moving Oracle table definitions and data to an external file for importing into another Oracle database.

The external tables feature and both drivers are integrated into the database server as extensions to the `CREATE TABLE SQL` statement. Thus, the external tables feature is readily available through any program that issues Oracle SQL, including `SQL*Plus` as

well as user-written applications. Refer to *Oracle Database Utilities* for more information about external tables and the associated access drivers.

On z/OS, all files accessed by an Oracle database server as part of the external tables feature must be HFS files. This includes any existing file you plan to use as a row source for an ORACLE_LOADER external table as well as all files written by the feature including log files, bad, and discard files. Like most other server file features, external tables use a database directory object as the locus of security. Although directory objects on Oracle Database for z/OS are capable of designating a data set name prefix, only directory objects that specify an HFS directory can be used with external tables. If you need to load Oracle database tables from a z/OS data set, you must use the regular SQL*Loader utility.

Datapump Export and Import

Both Datapump Export and Import, described in [Chapter 4, "Oracle Tools and Utilities on z/OS"](#) and the DBMS_DATAPUMP PL/SQL package are limited to processing HFS files on z/OS. Unique to z/OS, the file name values supplied by Datapump clients to an Oracle database server must begin with a "./" (dot-slash) prefix so they are clearly distinguished as HFS names. In addition, the directory object used in any z/OS Datapump operation must specify a valid HFS directory path.

Developing Oracle Applications to Run on z/OS

This chapter provides z/OS-specific information about design and runtime considerations for applications that will run in batch, TSO, or a shell. It also covers how to build an application: running Oracle Precompilers and OTT (Oracle Type Translator), compiling, and linking or binding a program. This includes both traditional (TSO/batch) mechanisms as well as a shell, including make. Most of the precompiler application build information in this chapter also pertains to applications that will run in CICS TS or IMS TM. Application design and runtime considerations and a few specific build considerations for CICS TS and IMS TM are covered at the end of this chapter.

Before reading this chapter, you should be familiar with [Chapter 2, "Oracle Software Interaction with z/OS and Language Environment."](#) Regardless of the language of your application, it will be using the Oracle program interface code, and is subject to features and mechanisms described there. In addition, you should review [Chapter 3, "Oracle Net and Server Connections on z/OS,"](#) which describes the mechanisms involved in connecting to an Oracle database server both locally and over a real network.

This chapter contains the following sections:

- [Overview](#)
- [Application Design Considerations](#)
- [Building an Application](#)
- [Building a Traditional Load Module with the Alternate API Stub](#)
- [Application Runtime Considerations](#)
- [Developing Oracle Applications for the CICS TS Environment](#)
- [Developing Oracle Applications for the IMS TM Environment](#)

Overview

Three facilities are provided for developing compiled high level language programs that run on z/OS and access an Oracle database: Oracle Precompilers, OCI, and OCCI.

Oracle Precompilers-Pro*C/C++, Pro*FORTRAN, Pro*COBOL, and Pro*PL/I-translate special EXEC SQL statements imbedded in a source program into language-specific declarations and executable statements that carry out the requested operation in a target Oracle database server. The output of the precompiler is compiled with a supported z/OS language compiler and then linked or bound as a

z/OS program object, conventional load module, or executable file in an HFS. Precompiler applications can run in batch, TSO, or a shell, or under control of CICS TS or IMS TM.

With OCI and OCCI, direct calls to defined Oracle API functions are coded in a C (OCI only) or C++ language program. The program is compiled with a supported z/OS C/C++ compiler and then linked or bound as a z/OS program object, conventional load module (OCI only) or executable file in an HFS. While more complex to use, OCI and OCCI provide access to virtually all Oracle features with fine-grained control over server operations. OCI and OCCI applications on z/OS can run in batch, TSO, or a shell. They are not supported in CICS TS or IMS TM environments.

All precompiler runtime, OCI and OCCI functions reside in the LIBCLNTS and LIBOCCI program objects that are part of the Oracle distribution. In a POSIX environment, LIBCLNTS and LIBOCCI are loaded from an HFS directory in LIBPATH. In a non-POSIX environment, LIBCLNTS and LIBOCCI are loaded from the JOBLIB/STEPLIB concatenation. Both LIBCLNTS and LIBOCCI are accessed through or in coordination with the Oracle stubs discussed below. For more information about LIBCLNTS and LIBOCCI and how they are accessed, refer to the section "[Application Runtime Considerations](#)" on page 6-29 and to [Chapter 2, "Oracle Software Interaction with z/OS and Language Environment."](#)

Precompiler, mixed precompiler/OCI applications and non-POSIX OCI applications must use an API stub. However, OCI POSIX applications can use an API stub. The API stub is used to resolve the application's references to precompiler and OCI functions. Traditional (non-XPLINK) linkage must be used. No other form of linkage to Oracle code is supported. All the API stubs except ORASTBX must be included at link or bind time; COBOL DYNAM applications include ORADMYC at link or bind time which loads ORASTBX at run time. Oracle applications that use COBOL DYNAM are subject to limitations. Refer to the "[Pro*COBOL Considerations](#)" section on page 6-12 for details on how to compile and link programs using the DYNAM option. For more information about binding and linking with API stubs, refer [Table 6-1](#).

Table 6-1 Binding and Link-Editing with API Stubs

Platform	Language	Bound with	Creates	Must Specify
Pro*C/C++ or OCI or mixed	ISO C	Binder	PM3/PM4	INCLUDE SQLLIB(ORASTBL)
	ISO C++	Binder	PM3/PM4	INCLUDE SQLLIB(ORASTBL)
	ISO C	Prelinker/Linkage Editor	PM2/Load Module	INCLUDE OBJLIB(ORASTBS)
	Pre-ISO C++	Prelinker/Linkage Editor	PM2/Load Module	INCLUDE OBJLIB(ORASTBS)
	ISO C (OCI V7 names, NOLONGNAME option)	Linkage Editor	PM2/Load Module	INCLUDE OBJLIB(ORASTBS)
	ISO C	c89	PM3/PM4	\$ORACLE_HOME/lib/orastbl.o
	ISO C++	c++	PM3/PM4	\$ORACLE_HOME/lib/orastbl.o
Pro*COBOL	COBOL	Binder	PM3/PM4	INCLUDE SQLLIB(ORASTBL)

Table 6–1 (Cont.) Binding and Link-Editing with API Stubs

Platform	Language	Bound with	Creates	Must Specify
	COBOL	Linkage Editor	PM2/Load Module	INCLUDE OBJLIB(ORASTBS)
	COBOL	Linkage Editor (DYNAM)	PM2/Load Module	INCLUDE OBJLIB(ORADMYC)
	COBOL	cob2	PM3/PM4	\$ORACLE_HOME/lib/orastbl.o
Pro*FORTRAN	FORTRAN	Linkage Editor	PM2/Load Module	INCLUDE OBJLIB(ORASTBF)
Pro*PL/I	PL/I	Binder	PM3/PM4	INCLUDE SQLLIB(ORASTBL)
	PL/I	Linkage Editor	PM2/Load Module	INCLUDE OBJLIB(ORASTBS)

OCCI and mixed OCI/OCCI applications must use XPLINK linkage and a DLL stub. However, OCI POSIX applications can use a DLL stub with traditional or XPLINK linkage. References to OCI and OCCI functions are implicitly resolved at bind time using the LIBCLNTS.x (client) and LIBOCCI.x (OCCI) side decks. No other form of linkage to Oracle code is supported. For more information about binding with DLL stubs, refer [Table 6–4](#).

Without exception, all precompiler, OCI, and OCCI applications on z/OS use LE services at runtime. You must use an LE-enabled compiler to build your applications and LE runtime services must be available when the application executes. In complex application designs—those involving multiple separate executables, multiple compiled languages, multiple tasks or threads, and similar techniques—you must adhere to requirements imposed by LE as well as ones imposed by Oracle.

If you are exploiting Oracle object-relational features in a Pro*C/C++, OCI, or OCCI application, you may also want to use OTT, a utility program that generates C language structure declarations corresponding to database object types.

Most of the Precompiler application build information in this chapter also applies to applications that will run in CICS TS or IMS TM. Additional details and requirements for CICS TS and IMS TM applications are provided later in this chapter.

Application Design Considerations

The foremost determiner of application design considerations is the intended execution environment. CICS TS and IMS TM have very specific requirements and limitations covered separately at the end of this chapter. The other traditional MVS environments (batch job and TSO) have one set of considerations while z/OS UNIX shells have another. In batch and TSO environments the LE POSIX attribute of your application can be either OFF or ON, as you choose, and this affects various LE behaviors such as interpretation of ambiguous filespecs. If your application runs in a shell, the LE POSIX attribute will always be ON.

Regardless of execution environment, your application's ultimate physical form will be one of several z/OS executable formats: a traditional load module (stored in a PDS), or a program object produced by the z/OS binder and stored in a PDSE or as an executable file in an HFS. In general you use the HFS for programs that will run in a shell and a PDS or PDSE for programs that run in other environments.

In terms of requirements, we can distinguish between simple and complex applications. A simple application meets all of the following criteria:

- Single load module or program object that does not dynamically invoke any other Oracle-accessing program, including Oracle tools and utilities
- Written in a single high level language
- Executes as a single batch job step, TSO command, or shell command
- Does not require z/OS APF authorization, run in supervisor state or a PSW key other than 8, or use an ASC mode other than primary

If your application fits the prior description, you can skip the section "[Requirements for Complex Applications](#)" on page 6-5.

In sections that follow, the term application context is used to refer to the collection of visible and internal state data representing the Oracle interactions of a single coherent application. An application context typically represents a connection to an Oracle database server and multiple SQL operations (cursors). It may include multiple connections to the same or different Oracle database servers, using specific facilities provided by OCI or an Oracle Precompiler for that purpose.

In addition, it is important to understand what LE calls an enclave in order to understand the requirements for z/OS Oracle application designs. An enclave is the set of resources and processing associated with a single logical LE application used within a single z/OS task (TCB), with an execution lifetime delineated by well-defined initialization and termination events. For more information about enclaves, refer to the *IBM Language Environment Concepts Guide* and related publications.

Basic Application Requirements

The following requirements apply to all Oracle-accessing applications that run on z/OS, from the simplest to the most complex:

- An application must use IBM Language Environment. In other words, you must use a compiler that generates LE-enabled code and that initializes an LE enclave when execution begins.
- An application can use any residency mode (RMODE) and either 31-bit or 24-bit addressing mode (AMODE), as permitted by the compiler you use. An application must use the same AMODE on all calls to Oracle API functions. If your application uses 24-bit addressing (AMODE=24), all parameters passed to Oracle interface functions must be addressable with a 24-bit address. For example, they must reside in virtual memory below the 16 MB line. All applications that are compiled with the XPLINK compiler option are AMODE=31.
- An application can be reentrant or not, as you choose and as supported by the compiler and linkage you use. The API and DLL stubs and all dynamically loaded Oracle interface code are reentrant, but the application itself is not required to be reentrant. Certain OCI and OCCI facilities require that you compile your C or C++ program with the DLL compiler option, which dictates the RENT option. All applications that are compiled with the XPLINK compiler option are reentrant.
- An application must not use the LE exit CEEBXITA. An Oracle-specific version of this exit is provided in the API and DLL stubs and cannot be overridden or replaced. This exit is used to provide cleanup of Oracle-related resources when your application ends; its actions are transparent to your application.
- If an application uses any functions implemented in an XPLINK DLL, for example, OCCI, an XPLINK application DLL, the ISO C++ STL, and C++ standard I/O

library functions, it must use the XPLINK(ON) LE runtime option. For more information about the XPLINK(ON) LE runtime option, refer to the ["Using the XPLINK\(ON\) LE Runtime Option"](#) section.

Requirements for Complex Applications

Read this section if your application does not meet one or more of the criteria for simple applications listed in the section ["Application Design Considerations"](#) on page 6-3, or if you are interested in more of the z/OS technical details involved in Oracle application execution. This discussion assumes familiarity with LE and other z/OS features and interfaces.

Using z/OS Assembler Language

Some complex application designs involve programming in z/OS Assembler Language and using low-level system services provided as Assembler macros or callable services. While you cannot use an Oracle Precompiler or make OCI or OCCI calls directly from Assembler, you can invoke Oracle-accessing C, C++, COBOL, FORTRAN, or PL/I programs from an Assembler program. Doing so requires, foremost, an understanding of LE and the notion of an LE enclave.

An Assembler program that invokes Oracle-accessing programs can be LE-conforming, using LE macros such as CEEENTRY and CEETERM for entry and exit linkage, or it can be a non-LE program. Which you use depends on what kind of Oracle-accessing programs are invoked and how, as discussed in the sections that follow.

Dynamic Linkage Techniques

There are a variety of ways in z/OS to cause program code to be read into memory and executed on demand of the initial or main application (the program invoked as a jobstep, TSO command, or shell executable). Some, such as DLLs, COBOL dynamic linkage, and the C functions `system()` and `fetch()`, are provided and supported by compilers and LE. Others, such as the Assembler services LOAD/CALL, LINK, and ATTACH (and related services such as ISPF's ISPLINK) operate outside LE. All of these techniques, and especially the latter category, require awareness of both LE requirements and Oracle requirements.

Using DLLs If permitted by the language and compiler, the application can exploit the DLL mechanism of LE. Your Oracle-accessing programs can call subroutines that are packaged in a DLL and the called subroutines themselves (in the DLL) can also be Oracle-accessing. When using DLLs, your main program must include the API or DLL stub, even if it does not contain any Oracle API calls. In an application that uses a DLL stub, only the main program in an LE enclave should contain the DLL stub.

In an application that uses an API stub, the Oracle API function calls in a DLL always go through the API stub included in the DLL. In an application that uses a DLL stub, the Oracle API function calls in a DLL are implicitly resolved by binding the DLL against the client and OCCI side decks.

When you use DLLs, both the subroutines in the DLL and any Oracle API functions execute in the caller's LE enclave.

Using the XPLINK(ON) LE Runtime Option If any application code is compiled with XPLINK or uses a DLL that is compiled with XPLINK, such as OCCI, the ISO C++ STL, the ISO C++ standard I/O library, the main program must use the XPLINK(ON) LE runtime option. This can be done by using one of the following methods:

- Compiling the main program with XPLINK, where permissible.
- Compiling the main program with `#pragma runopts(XPLINK(ON))`.
- Placing `XPLINK(ON)` into the `_CEE_RUNOPTS` environment variable. This is the only supported method for COBOL and PL/I main programs.

For more information, refer to *z/OS C/C++ Programming Guide* or *z/OS Language Environment Programming Guide*.

Using COBOL Dynamic Linkage Your Pro*COBOL application can use COBOL dynamic linkage to invoke your own subroutines. As with DLLs, your main program must include the API stub regardless of whether it contains Oracle API calls. Subroutines invoked with COBOL dynamic call linkage and containing Oracle API calls must also include the API stub.

Oracle also allows you to use the COBOL DYNAM compiler option, subject to certain limitations. When you compile with DYNAM, the Pro*COBOL API calls themselves are treated as dynamic COBOL calls, loading a special runtime version of the API stub.

When you use COBOL dynamic linkage, the called subroutines execute in the caller's LE enclave.

Using C/C++ `system()` The `system()` function of C/C++ library can be used to invoke a batch program, a TSO command, CLIST, or EXEC, or a shell command or script. From an LE standpoint, a program or script invoked in this fashion is not a subroutine; it is a main program and runs in its own LE enclave. This means your application is using multiple LE enclaves, discussed in the section "[Multiple LE Enclaves in an Application](#)" on page 6-7. You cannot pass Oracle connection state data (such as cursor or connection handles) to a program invoked using `system()`. Programs invoked with `system()` must establish their own independent interactions with an Oracle server. You can use `system()` to invoke Oracle tool or utility programs, such as SQL*Plus.

Using C/C++ `fetch()` The `fetch()` function of C/C++ library loads an executable, returning a function pointer that can be used to invoke the executable one or more times. A program invoked this way runs as a subroutine in the caller's LE enclave and cannot be a main routine. If you use `fetch()` to run one or more Oracle-accessing subroutines, you must include the API stub in the main program and in all Oracle-accessing subroutines or include the DLL stub in the main program and bind all Oracle-accessing routines against the client and OCCI side decks.

Using z/OS LOAD/CALL or LINK These services are accessed directly using z/OS Assembler Language macros. The LINK service is also used internally by other facilities, including ISPLINK under TSO ISPF. These mechanisms operate outside of LE, so care must be taken to ensure that LE facilities are not impacted.

If you use LOAD and CALL to invoke an Oracle-accessing program, one of two conditions must be met when CALL is issued:

- No LE enclave exists at the time of the CALL, and the called program operates as a main program, initializing its own LE enclave and terminating it before returning.
- An LE enclave exists, the Assembler program that issues the CALL is LE-conforming, and the called program operates as a subroutine and runs in the same enclave-it cannot be a main routine. In this case, the main program must include the API or DLL stub, even if it is written entirely in Assembler and contains no Oracle API calls.

When you use a LINK macro to invoke an Oracle-accessing program, the called program must be a main routine that initializes its own LE enclave. If an enclave

already exists at the time of the LINK, the called program runs as a nested enclave, discussed in the section "[Multiple LE Enclaves in an Application](#)" on page 6-7. Regardless of which condition exists, you cannot pass Oracle connection state data, such as cursor or connection handles, to the called program.

Using z/OS ATTACH The ATTACH macro creates a subtask, which means you are using multitasking, discussed in the section "[Multiple LE Enclaves in an Application](#)" on page 6-7. The program you run with ATTACH must be a main routine that creates its own LE enclave. You cannot pass Oracle connection state data, such as cursor or connection handles, from one z/OS task to another.

Multiple LE Enclaves in an Application

There are a couple of ways that an application that runs as a single z/OS task (TCB) can use multiple LE enclaves. One is for a program in an existing enclave to invoke another main program using a service such as C/C++ `system()` or Assembler LINK. In this case LE creates what is called a nested enclave for the invoked program. The nesting can go deeper if the invoked program invokes still other main programs the same way.

Another scenario is when a non-LE Assembler program invokes multiple LE main programs in succession using LOAD/CALL or LINK. Each invoked program creates and ultimately deletes its own LE enclave. Since there is no existing enclave at create time, these enclaves are not nested, and only one enclave exists at a time.

Regardless of which scenario applies (or both, since they can be combined), only one LE enclave is active (running program code) at a time. When the enclaves involve Oracle interaction, when the programs are precompiler, OCI, or OCCI applications, the following considerations apply:

- A single Oracle application context, as defined earlier, can be created and used within a single LE enclave only. This means your application cannot be designed to pass resources from one application context (such as cursors or OCI handles) to code that executes in a different LE enclave, including a nested enclave. In practical terms, this prohibits opening a cursor in one COBOL run unit, which equates to an LE enclave, and trying to fetch from that cursor in a different COBOL run unit. It also prohibits passing cursors or other Oracle connection state artifacts from one C/C++ main program to another C/C++ main program.
- If an application uses multiple LE enclaves with API stubs, all of the precompiler or OCI programs making up the application must have been built and bound with the API stub from either Oracle Database 10g or Oracle9i Release 2. Applications built with the Oracle9i API stub must use a version of the stub that contains the fix for Oracle bug 3431417.
- If an application uses multiple LE enclaves with DLL stubs, the main program in each enclave that makes Oracle API calls must be built and bound with a DLL stub shipped with Oracle Database 10g release 2. Any main programs or application DLLs that make Oracle API calls must be bound against the client and OCCI side decks.
- Pro*COBOL applications that use the COBOL DYNAM compile option and, therefore, do not include the API stub, cannot be used in a multiple enclave application.

Multitasking Applications

Using Assembler Language and the ATTACH macro and related services, you can develop an Oracle application that uses multiple concurrently dispatchable tasks. Both

Oracle and LE impose requirements on such designs. For LE, enclaves normally have task scope: an enclave created in one task cannot be used to execute programs in another task. For Oracle, all connection state data (the application context) also has task scope. You cannot pass connection state data such as cursor or connection handles from one z/OS task to another.

These requirements essentially mean that each z/OS task in a multitasking design must be independent of the other tasks as far as LE and Oracle interactions are concerned. The tasks are otherwise free to communicate and synchronize among themselves. Such designs can be complex and difficult to debug.

z/OS Environment and z/Architecture Hardware States

In z/OS there are numerous combinations of software and hardware states in which a program can run. This includes variations of dispatchable unit (task or SRB), APF authorization, supervisor versus problem state, PSW key, and addressing variations including AR and cross-memory modes. Your Oracle-accessing application is constrained to using environments and execution states that are supported by LE. In addition, the following restrictions apply:

- You cannot access Oracle when running as an SRB; only task mode is supported.
- Your program must be in primary ASC mode and not in any cross-memory mode at the time an Oracle interface call is made.
- Your program must not be disabled, hold any z/OS lock, or have an Enabled Unlocked Task (EUT) FRR established at the time of an Oracle interface call.
- All input and output parameters for an Oracle interface call must be accessible in your application's address space using the application's AMODE and PSW key.
- All calls to Oracle from an application must be made in the same AMODE, the same privilege state (problem or supervisor), and the same PSW key.
- If your application uses PROTOCOL=XM to connect to a local Oracle server, z/OS Asynchronous Exits (IRBs) are not dispatched during Oracle server interaction (while running in the Oracle server in cross-memory mode). This means processes like z/OS STIMER/STIMERM timer pops and TSO attention interrupts are deferred until a server call returns control.

POSIX Threading

You can use the POSIX threading features of z/OS in an Oracle Pro*C/C++, OCI or OCCI application. However, because z/OS normally dispatches separate threads on separate z/OS tasks, you cannot pass Oracle connection state data (such as connection or cursor handles) from one thread to another. Each thread must establish, use, and close its own Oracle server connections and cursors.

OCI Interface to Publish/Subscribe

The OCI interface to Oracle's publish/subscribe features allows an application to register a callback function that is driven when there are queue data or trigger events to process. This feature is supported with Oracle Database for z/OS as long as the client executes in a multi-threaded environment and there is TCP connectivity between the server and the client. This includes local z/OS clients running under z/OS UNIX Systems Services shell environments as well as remote clients on any multi-threaded platform. It excludes native TSO/batch and IMS TM clients at this time

The TCP connectivity requirement does not mean that the client must be connected to the server using Oracle Net PROTOCOL=TCP. The client can be connected using any protocol (including Oracle Net PROTOCOL=XM for a local z/OS UNIX Systems

Services client) and in fact the client can disconnect its session from the database after setting up the callback without affecting the callback. A special background process in the server (EMON) opens a TCP connection to the client thread that gets created when `OCISubscriptionRegister` is called. TCP communication from the server to the client thread ultimately drives the callback functions.

Refer to the *OCI Programmer's Guide* for further details on the OCI interface to Oracle's publish/subscribe facility.

Building an Application

The following steps outline how to build an application to access an Oracle database using API stubs:

1. Precompile your program with one of the Oracle Precompilers and other precompilers that are required by your target environment (not required for OCI applications).
2. Compile your program with the host language compiler.
3. Link your program with an API stub designed especially for the target environment.
4. Run your program in the target environment.

The following steps outline how to build an application to access an Oracle database using DLL stubs:

1. Compile your program with the host language compiler.
2. Bind your program with a DLL stub designed especially for the target environment.
3. Run your program in the target environment.

Precompiling Programs

Oracle Precompilers are used to process C/C++, COBOL, FORTRAN, and PL/I programs before passing them to their native compilers. They translate embedded SQL statements into the appropriate language statements and calls necessary to access an Oracle database. Oracle Precompilers are supported in all three operating environments (batch, TSO, and shell) except Pro*FORTRAN and Pro*PL/I, which are not supported in the shell environment.

You should review the information in the following product-specific documentation on Oracle Precompilers before proceeding with the z/OS-specific information in this section:

- *Programmer's Guide to the Oracle Precompilers*
- *Pro*C/C++ Precompiler Programmer's Guide*
- *Pro*COBOL Precompiler Programmer's Guide*
- *Pro*FORTRAN Supplement to the Oracle Precompilers Guide*
- *Pro*PL/I Supplement to the Oracle Precompilers Guide*

Oracle Precompiler Executables

Oracle Precompilers can be run in various environments. The following table shows the names of the executables for each environment:

Table 6–2 Oracle Precompiler Executables

Precompiler	Batch/TSO Executable Name	JCL Procedure Name	POSIX Executable Name
Pro*C/C++	PROCI	ORAC	proc
Pro*COBOL	PROCOB	ORACOB	procob
Pro*FORTRAN	PROFOR	PFORCLGO	N/A
Pro*PL/I	PROPLI	ORAPLI	N/A

Oracle Precompiler INCLUDE Files

Several INCLUDE files are provided in the H data set for use by the Oracle Precompilers. You do not have to specify the member name that includes the language suffix in your source program. You may specify the data structure name, for example, SQLCA, in your source program. Oracle Precompilers include the appropriate member name based on the Oracle Precompiler language, as listed in the following table:

Table 6–3 Oracle Precompiler INCLUDE Files

Language	Member Name	Data Structure
C/C++	SQLCAC	SQLCA
C/C++	ORACAC	ORACA
C/C++	ORADAC	ORADA
COBOL	SQLCACOB	SQLCA
COBOL	ORACACOB	ORACA
FORTRAN	SQLCAFOR	SQLCA
FORTRAN	SQLCAFOR	ORACA
PL/I	SQLCAPLI	SQLCA
PL/I	ORACAPLI	ORACA

For more information about data structures, refer to your Oracle Precompiler documentation.

Oracle Precompiler Options

Because there are many possible options for the Oracle Precompilers, you might find it impossible to fit all the options you need in the 100 bytes z/OS allows for parameters initially passed to user programs (for example, from JCL). You can use the following methods to pass more than 100 bytes of precompiler options to an Oracle Precompiler interface:

- Use the Oracle Precompiler EXEC ORACLE OPTION statement in your source program to specify some of the parameters. This is a good way to specify options that do not change for each compile. Refer to your Oracle Precompiler documentation for information about the EXEC ORACLE OPTION statement.
- Use a parameter file to direct the Oracle Precompiler to retrieve parameters from a z/OS file in addition to the z/OS parameter field. Refer to [Chapter 2](#) for more information about the parameter file feature. A combination of option overrides and parameter file also can be used. For example:

```
PARM='++//DD:ddname option1 option2...'
```


- When using a question mark to determine the current value for one or more options, surround the question mark with full quotes or escape the question mark with a backslash, as shown in the following examples:

```
// EXEC PGM=PROC, PARM=' "?" '
// EXEC PGM=PROC, PARM=' \? '
```

When specifying Oracle Precompiler options on the command line, separate each option with one or more blanks. Do not use a comma as a separator. To specify a fully-qualified data set name in an Oracle Precompiler option, use the appropriate LE syntax.

When running in a batch or TSO environment, some Oracle Precompiler options require the following special considerations. Refer to [Chapter 2](#) for rules on how to specify z/OS data set names to the Oracle database server.

INAME specifies the input file containing the source code, in any supported language, for the precompiler. The INAME file can be a sequential data set, a PDS member, an instream (DD *) data set, or an HFS file. In batch and TSO environments there is no default. Typically, this operand is coded as shown in the following example with a SYSIN DD statement pointing to the input source code:

```
INAME=//DD:SYSIN
```

ONAME specifies the file to contain the output from the Oracle Precompiler. The ONAME file can be a sequential data set, a PDS member, or an HFS file. In batch and TSO environments there is no default. Typically, this operand is coded as shown in the following example with a SYSPUNCH DD statement pointing to a data set that is passed as input to the native language compiler:

```
ONAME=//DD:SYSPUNCH
```

LNAME specifies the file to contain the listing from the Oracle Precompiler. The LNAME file can be a sequential data set, a PDS member, a SYSOUT data set, or an HFS file. In batch and TSO environments there is no default. The LNAME parameter cannot specify the SYSPRINT DD. Typically, this operand is coded as shown in the following example with a LIST DD statement pointing to a SYSOUT data set:

```
LNAME=//DD:LIST
```

Configuration Files

Oracle Precompilers on z/OS support the concept of system and user configuration files. In batch and TSO environments, the system configuration file is read from the PROCFG DD statement, if it exists. To avoid a warning message, the PROCFG DD statement should be DUMMY if no changes to the default options are desired.

The user configuration file can be used to make overrides to the system configuration file for specific options. The file must be specified in the parameters passed to the precompiler using the CONFIG= option. The CONFIG= option is not allowed inside a configuration file; doing so will produce an error message, although the precompile continues. If a data set is used to pass precompiler options to an Oracle Precompiler, the data set must not have sequence numbers. If sequence numbers are found, then the Precompiler stops processing.

When using configuration files in the POSIX environment, follow the rules outlined in the product documentation.

In all environments, each option in a configuration file must be on a separate line and must begin in column 1.

Return Codes

Oracle Precompilers set a return code of 0 upon successful precompilation of source code. A return code of 4 is set if the precompiler issues a warning message.

A return code of 8 is set if the precompiler detects a potentially irrecoverable error condition. In this case, an error message is written to the file specified in the LNAME parameter passed to the precompiler.

Language-Specific Coding Considerations

The following sections discuss certain considerations that impact Oracle Precompiler programs written in a particular language, regardless of the target environment running the program:

Compiler Support Considerations Oracle Corporation supports any currently supported IBM compiler that uses the LE runtime environment.

Pro*COBOL Considerations When using Pro*COBOL, each of the following has special considerations:

- **RETURN-CODE Special Register.** On z/OS, COBOL programs use the RETURN-CODE special register to pass a return code from a main program back to z/OS. It is then used to form the job step completion code. The z/OS standard subroutine linkage places unpredictable values into the RETURN-CODE special register as a subroutine is called. Subprograms on z/OS can set the value of the RETURN-CODE special register as they exit. However, the RETURN-CODE special register is an IBM extension to the COBOL language and is not part of the SQL standard.

In compliance with the SQL standard, Oracle Pro*COBOL does not make use of the RETURN-CODE special register and does not explicitly set the value. This causes the value of the RETURN-CODE special register to be unpredictable after each SQL statement completes. If a SQL statement is issued immediately before the main program returns to z/OS, then the unpredictable value remains in the RETURN-CODE special register and is used to form the job step completion code. The application developer is responsible for ensuring the correctness of the RETURN-CODE special register.

- **INTEGER Values.** In the Oracle product documentation, SQL library function parameters are often documented with a data type of INTEGER. In z/OS, integers are a fullword and must be defined in COBOL as fullword COMP fields with sizes from S9(5) to S9(8). An S9(4) or smaller definition generates a halfword that is the default integer size on some operating systems. This can cause problems when migrating applications from other operating systems to z/OS. For example, a call to SQLIEM expects an integer MESSAGE_LENGTH as the second parameter. Define a MESSAGE_LENGTH field as PIC S9(8) COMP in your program. A call to SQLGLM expects the second and third parameters, BUFFER_SIZE and MESSAGE_LENGTH respectively, to be integers. As with SQLIEM, they must be defined as PIC S9(8) COMP fields.
- **DYNAM Compiler Option.** Pro*COBOL support for the DYNAM option is provided with a loadable version of the API stub. This support is meant for installations whose standards or existing application designs require the use of DYNAM; it is not recommended for general use. When you use COBOL DYNAM, the Oracle LE exit CEEBXITA is not included with your application and does not

run at enclave termination. This means Pro*COBOL programs that use DYNAM support cannot be executed as part of a multiple LE enclave design. Refer to the sections "[Compiling Programs](#)" on page 6-17, "[Linking Programs](#)" on page 6-18, and "[Oracle Interface Initialization, Processing, and Error Handling](#)" on page 6-30 for details on using COBOL DYNAM support.

Pro*C/C++, OCI, and OCCI Considerations When using Pro*C/C++, OCI, or OCCI, object support has special considerations. The OTT utility is used to convert database object type definitions into C programming language declarations that are included in your Pro*C/C++, OCI, or OCCI applications. OTT is a Java application which must run in a POSIX environment. For more information about OTT, refer to your Oracle Precompiler documentation.

All Oracle interface code is compiled with the ENUMSIZE(INT) option. In order to be compatible, your Pro*C/C++, OCI, and OCCI programs must also use ENUMSIZE(INT).

Currently only FLOAT(HEX) is supported in Pro*C/C++, OCI, and OCCI programs on z/OS. You cannot use FLOAT(IEEE) in an Oracle-accessing application.

Pro*FORTRAN Considerations Pro*FORTRAN is supported in the non-POSIX batch and TSO environments only. It is not supported in the POSIX environment.

Pro*PL/I Considerations When using Pro*PL/I, the record format of the output file and the execution environment have special considerations, as follows:

- The native language output from the Pro*PL/I Precompiler that is directed to the data set specified in the ONAME parameter cannot have a RECFM of V. This is because a z/OS compiler restriction requires all PL/I programs contained in a variable length data set to have a blank or a sequence number in columns 1 to 8. This restriction does not apply to the data set input to Pro*PL/I, only to the intermediate data set passed to the PL/I compiler.
- Pro*PL/I is supported in the non-POSIX batch and TSO environments only. It is not supported in the POSIX environment.

Special Considerations for Running Precompilers in Batch and TSO Environments

The Oracle Precompilers use temporary files during their processing. In the batch and TSO environments, users must provide DD statements SYSUT1, SYSUT2, SYSUT3, SYSUT4 (Pro*FORTRAN only) and SYSUT5 (Pro*FORTRAN only) that point to temporary files. These are normally specified as UNIT=VIO DD statements.

If the ORECLen precompiler parameter is larger than 132, then the DCB attributes must be specified in these DD statements to set the LRECL equal to the ORECLen value. For example, the following statement sets the LRECL to 200:

```
//SYSUT1 DD UNIT=VIO,DCB=(LRECL=200)
```

Sample JCL for Running the Pro*C/C++ Precompiler in Batch Environments

The following sample JCL illustrates the precompile step necessary to build a Pro*C precompiler application program:

```
//PRECOMPL EXEC PGM=PROC,
// PARM='++//DD:SYSPARM'
//STEPLIB DD DSN=ORACLE.V10G.CMDLOAD,DISP=SHR
//ORA$LIB DD DSN=ORACLE.V10G.MESG,DISP=SHR
//SYSLIB DD DSN=ORACLE.V10G.H,DISP=SHR
// DD DSN=SYS1.SCEEH.H,DISP=SHR
```

```

//          DD DSN=SYS1.SCEEH.SYS.H,DISP=SHR
//LIST     DD SYSOUT=*
//PROCFG  DD DUMMY
//SYSIN   DD DISP=SHR,DSN=JSMITH.CPGM.PC
//SYSPUNCH DD DSN=&&PCCOUT,DISP=(,PASS,DELETE),
//          UNIT=VIO
//SYSPARM DD *
INAME=//DD:SYSIN
ONAME=//DD:SYSPUNCH
LNAME=//DD:LIST
/*
//SYSUT1 DD UNIT=VIO
//SYSUT2 DD UNIT=VIO
//SYSUT3 DD UNIT=VIO

```

Sample JCL for Running the Pro*COBOL Precompiler in Batch Environments

The following sample JCL illustrates the precompile step necessary to build a Pro*COBOL precompiler application program:

```

//PRECOMPL EXEC PGM=PROCOB,
// PARM='++//DD:SYSPARM'
//STEPLIB DD DSN=ORACLE.V10G.COMDLOAD,DISP=SHR
//ORA$LIB DD DSN=ORACLE.V10G.MESG,DISP=SHR
//SYSLIB DD DSN=ORACLE.V10G.H,DISP=SHR
//          DD DSN=SYS1.SCEEH.H,DISP=SHR
//          DD DSN=SYS1.SCEEH.SYS.H,DISP=SHR
//LIST     DD SYSOUT=*
//PROCFG  DD DUMMY
//SYSIN   DD DISP=SHR,DSN=JSMITH.COBPGM.PCO
//SYSPUNCH DD DSN=&&PCCOUT,DISP=(,PASS,DELETE),
//          UNIT=VIO
//SYSPARM DD *
INAME=//DD:SYSIN
ONAME=//DD:SYSPUNCH
LNAME=//DD:LIST
/*
//SYSUT1 DD UNIT=VIO
//SYSUT2 DD UNIT=VIO
//SYSUT3 DD UNIT=VIO

```

Sample JCL for Running the Pro*FORTRAN Precompiler in Batch Environments

The following sample JCL illustrates the precompile step necessary to build a Pro*FORTRAN precompiler application program:

```

//PRECOMPL EXEC PGM=PROFOR,
// PARM='++//DD:SYSPARM'
//STEPLIB DD DSN=ORACLE.V10G.COMDLOAD,DISP=SHR
//ORA$LIB DD DSN=ORACLE.V10G.MESG,DISP=SHR
//SYSLIB DD DSN=ORACLE.V10G.H,DISP=SHR
//LIST     DD SYSOUT=*
//PROCFG  DD DUMMY
//SYSIN   DD DISP=SHR,DSN=JSMITH.FORPGM.FOR
//SYSPUNCH DD DSN=&&PCCOUT,DISP=(,PASS,DELETE),
//          UNIT=VIO
//SYSPARM DD *
INAME=//DD:SYSIN
ONAME=//DD:SYSPUNCH
LNAME=//DD:LIST
/*

```

```
//SYSUT1 DD UNIT=VIO
//SYSUT2 DD UNIT=VIO
//SYSUT3 DD UNIT=VIO
//SYSUT4 DD UNIT=VIO
//SYSUT5 DD UNIT=VIO
//PRECOMPL EXEC PGM=PROCOB,
```

Sample JCL for Running the Pro*PL/I Precompiler in Batch Environments

The following sample JCL illustrates the precompile step necessary to build a Pro*PL/I precompiler application program:

```
//PRECOMPL EXEC PGM=PROPLI,
// PARM='++//DD:SYSPARM'
//STEPLIB DD DSN=ORACLE.V10G.CMDLOAD,DISP=SHR
//ORA$LIB DD DSN=ORACLE.V10G.MESG,DISP=SHR
//SYSLIB DD DSN=ORACLE.V10G.H,DISP=SHR
//          DD DSN=SYS1.SCEEH.H,DISP=SHR
//          DD DSN=SYS1.SCEEH.SYS.H,DISP=SHR
//LIST DD SYSOUT=*
//PROCFG DD DUMMY
//SYSIN DD DISP=SHR,DSN=JSMITH.PLIPGM.PPL
//SYSPUNCH DD DSN=&&PCCOUT,DISP=(,PASS,DELETE),
//          UNIT=VIO
//SYSPARM DD *
INAME=//DD:SYSIN
ONAME=//DD:SYSPUNCH
LNAME=//DD:LIST
/*
//SYSUT1 DD UNIT=VIO
//SYSUT2 DD UNIT=VIO
//SYSUT3 DD UNIT=VIO
```

Sample Commands for Running Oracle Precompilers in a Shell

Oracle Precompilers running in a shell must know the include pathname to find all the include files. Therefore, the command line must have each nonstandard include pathname specified.

The following are sample commands for running Oracle Precompilers in a shell:

Pro*COBOL

```
$ procob iname=mycobpgm.pco include=/home/jones/copybooks oname=xyz.cbl
```

Pro*C/C++

```
$ proc iname=mycpgm.pc include=/home/jones/include
```

Compiler Options for Oracle Applications

This section contains information regarding compiler options for Oracle applications. The settings are grouped by language with an environment indicator to reflect whether the setting applies.

Environment indicators are listed as follows:

C - CICS TS client

I - IMS TM client

N - Native (non-POSIX) batch and TSO client with API stub

PA - POSIX client with API stub

PD - POSIX client with DLL stub

Z - OCCI native (non-POSIX) batch and TSO client with DLL stub

C/C++ Compiler Options

The C/C++ compiler options are listed as follows:

- **DLL - N, PA, PD, PZ**
Required for C if using OCI callbacks or a DLL stub. Required for C++
- **LANGLVL(EXTENDED) - N, PA, PD, Z**
Required if using OCI-Lob related functions or a DLL stub
- **ENUMSIZE(INT) - C, I, N, PA, PD, Z**
Required for OCI/OCCI programs
- **FLOAT(HEX) - C, I, N, PA, PD, Z**
Required (CHECK XPLINK!)
- **LONGNAME - N, P**
Default for C. Required for DLL stub applications. For an OCI API stub application, LONGNAME allows prelinking and linking with ORASTBS or linking with ORASTBL. Refer to the section "[Linking Programs](#)" on page 6-18 or more information.
- **NOLONGNAME - N, P**
For an OCI API stub application that uses OCI V7 names or the DEFINE=ORA_SNAME precompiler option, NOLONGNAME allows linking with ORASTBS. Refer to the "[Building a Traditional Load Module with the Alternate API Stub](#)" section on page 6-26 for more information.
- **TARGET(LE,zOSV1R4) - N, PA, PD, Z**
zOSV1R4 or later is required. Default on z/OS V1.4 or later
- **NOXPLINK - C, I, N, PA, PD**
Required C, I, N, PA. Optional for PD.
- **XPLINK - PD, Z**
Optional for PD. Required for Z. Implies DLL, LONGNAME.
- **SEARCH(\$ORACLE_HOME/rdbms/public) - PD, Z**
Required for OCCI

COBOL Compiler Options

The COBOL compiler options are listed as follows:

- **APOST/QUOTE - C, I, N, P**
The PCC option LITDELIM must match the value specified. For CICS TS, this value must also be specified as a CICS translator option.
- **NODYNAM/DYNAM - N, P**
NODYNAM is recommended. To use the DYNAM option, refer to the sections "[Compiling Programs](#)" on page 6-17, "[Linking Programs](#)" on page 6-18, and "[Oracle Interface Initialization, Processing, and Error Handling](#)" on page 6-30 for details on using COBOL DYNAM support.
- **PGMNAME - N, P**
LONGMIXED - requires linkage with ORASTBL.

LONGUPPER - requires linkage with ORASTBL.

Refer to the section "[Linking Programs](#)" on page 6-18 or more information.

- SQL - C, I, N, P
Not applicable for Oracle applications

PL/I Compiler Options

The PL/I compiler options are listed as follows:

- NOINTERRUPT - N,P
Required

- NOT,OR -

Refer to the *PRO*PL/I Supplement to the Oracle Precompilers Guide* for restrictions on using PL/I logical operators in SQL statements

Compiling Programs

Precompiled, OCI, and OCCI applications must be compiled using their host language compiler. In the case of precompiled applications, the output generated by the precompiler is used as input to the host language compiler.

Use of the DLL compiler option when compiling Pro*COBOL, Pro*C/C++, OCI, or OCCI programs is an option and is generally determined by the design of the application. This is supported for Oracle applications, but with two exceptions:

- Pro*COBOL applications compiled with the COBOL DYNAM option cannot specify the DLL option.
- OCI C API stub applications that make use of callbacks, C++ applications and DLL stub applications.

Considerations for building applications to run in the CICS TS and IMS TM environments are covered in the sections "[Developing Oracle Applications for the CICS TS Environment](#)" on page 6-33 and "[Developing Oracle Applications for the IMS TM Environment](#)" on page 6-35.

Sample Commands for Compiling Programs in a Shell

The following are sample commands for compiling programs in a shell:

COBOL

```
export STEPLIB=SYS1.SIGYCOMP
export COBOPT='QUOTE,OPTIMIZE,SOURCE,LIST'
cob2 -c pgm_name.cbl -o pgm_name.o -v \
-I. \
-I$ORACLE_HOME/precomp/public \
2>pgm_name.err
```

C

```
c89 -c \
-I. \
-I$ORACLE_HOME/rdbms/public \
-I$ORACLE_HOME/precomp/public \
-o pgm_name.o pgm_name.c \
2>pgm_name.err
```

Compiling OCCI programs to Run in Batch and TSO Environments

The following sample JCL, based on standard procedure CBCXCB, illustrates the compilation step necessary for an OCCI C++ program in batch and TSO environments.

```
//COMPILE EXEC PGM=CCNDRVR,REGION=96M,
//          PARM=(' /CXX XPLINK GOFF OPTFILE')
//SYSLIN DD DISP=(,PASS),UNIT=VIO,
//          DCB=(RECFM=FB,LRECL=80,BLKSIZE=3120)
//SYSIN DD - Refer to Note 1 -
//SYSPRINT DD SYSOUT=*
//SYSOUT DD SYSOUT=*
//SYSPRT DD SYSOUT=*
//SYSOPTF DD *
          LANGLVL(EXTENDED) ENUMSIZE(INT)
          SEARCH($ORACLE_HOME/rdbms/public) - Note 2 -
*/
```

Usage Notes:

1. Point to the program to be compiled. This can be in a data set (DD DSN) or an HFS file (DD PATH).
2. Replace \$ORACLE_HOME with the Oracle home directory.

Linking Programs

After a program has been compiled, the resulting object code is passed to the binder, where it is linked with the API stub.

Considerations for linking applications to run in the CICS TS and IMS TM environments are covered in the sections ["Developing Oracle Applications for the CICS TS Environment"](#) on page 6-33 and ["Developing Oracle Applications for the IMS TM Environment"](#) on page 6-35.

Linking Pro*C/C++, OCI, COBOL, and PL/I Programs to Run in Batch and TSO Environments

When linking a C, C++, COBOL, or PL/I program to run in batch or TSO, add the following linkage editor control statement to the SYSLIN DD statement:

```
INCLUDE SYSLIB(ORASTBL)
```

If there are unresolved external references for symbols whose names begin with SQL, then ensure ORASTBL is included in the linkedit. This API stub resolves any calls in the generated code. If ORASTBL is correctly included, then the problem is probably caused by a missing entry point to the API stub routine. You can contact Oracle Support Services for additional assistance.

The following sample JCL illustrates the linking step necessary to run C, C++, COBOL, and PL/I programs in batch and TSO environments:

```
//LKED EXEC PGM=IEWL,
// PARM='XREF,LET,LIST,DYNAM=DLL'
//SYSPRINT DD SYSOUT=*
//SYSLIB DD DSN=ORACLE.V10G.SQLLIB,DISP=SHR
//          DD DSN=SYS1.SCEELKED,DISP=SHR
//          DD *- Refer to Note 1 -*
//SYSLMOD DD *- Refer to Note 2 -*
//TEXT DD *- Refer to Note 3 -*
//SYSUT1 DD UNIT=VIO
//SYSLIN DD *
INCLUDE TEXT *- Refer to Note 3 -*
```



```
INCLUDE SYSLIB(ORASTBL)                *- Refer to Note 4 -*
                                        *- Refer to Note 5 -*
/*
```

Usage Notes:

1. Include additional libraries required by your program and any runtime libraries required by the native language.
2. This DD statement describes the library in which your linkedited Oracle load module is placed.
3. This DD statement points to the object output from your Oracle Precompiler program's native language compiler.
4. The specific contents of the SYSLIN DD statement depend on the target environment. Refer to "[Linking Programs](#)" on page 6-18 for more information.
5. Include any additional linkage editor statements you might require.

Linking COBOL Programs Using DYNAM

Each target environment has a unique stub, and different linking considerations apply to each of the target environments. In all cases, unless you are using the COBOL DYNAM compiler option, the API stub must be linked statically linked with your program.

If you are using the DYNAM option when building a Pro*Cobol application, you need to make the following changes for linking your program:

- Remove any DYNAM(DLL) parameter
- Add an OBJLIB DD statement allocating `oracle_hlq.OBJLIB`
- For link/binder control, replace the following control statement:

```
INCLUDE SQLLIB(ORASTBL)
```

Instead, use the following statement:

```
INCLUDE OBJLIB(ORADMYC)
```

If the application developer does not change or incorrectly changes the INCLUDE statement, it will probably go undetected when the application is built. The mistake will result in an all-too-common 0C4 abend. Tracing it back to the build error, the reason will not be apparent without a significant amount of debugging. To help with this problem, extra checking was added to the runtime interface to detect the condition before the 0C4 abend is likely to occur.

Detection that the LE initialization is insufficient for C, results in a standard API failure message, such as:

```
MIC011E Oracle API processing error, reason code 014
```

This is followed by a U2010 user abend with reason code 14. Reason code 14 uniquely identifies this as the build-time mistake: ORADMYC was not included.

Linking FORTRAN programs to Run in Batch and TSO Environments

When linking a FORTRAN program to run in batch or TSO, add the following linkage editor control statement to the SYSLIN DD statement:

```
INCLUDE SYSLIB(ORASTBF)
```

If there are unresolved external references for symbols whose names begin with SQL, then ensure ORASTBF is included in the linkedit. These API stubs resolve any calls in the generated code. If ORASTBF is correctly included, then the problem is probably caused by a missing entry point to the API stub routine. You can contact Oracle Support Services for additional assistance.

The following sample JCL illustrates the linking step necessary to run FORTRAN programs in batch and TSO environments. Note that SYSLMOD may be either a PDS (for a traditional load module) or PDSE (for a PM2 program object).

```
//LKED EXEC PGM=HEWL,
// PARM='LET,LIST,MAP,XREF'
//SYSPRINT DD SYSOUT=*
//SYSLIB DD DSN=SYS1.SCEELKED,DISP=SHR
// DD - Refer to Note 1 -
//OBJLIB DD DSN=ORACLE.V10G.OBJLIB,DISP=SHR
//SYSLMOD DD - Refer to Note 2 -
//TEXT DD - Refer to Note 3 -
//SYSUT1 DD UNIT=VIO
//SYSLIN DD *
INCLUDE TEXT - Refer to Note 3 -
INCLUDE OBJLIB(ORASTBF) - Refer to Note 4 -
- Refer to Note 5 -
/*
```

Usage Notes:

1. Include additional libraries required by your program and any runtime libraries required by the native language.
2. This DD statement describes the library in which your linkedit Oracle load module is placed.
3. This DD statement points to the object output from your Oracle Precompiler program's native language compiler.
4. The specific contents of the SYSLIN DD statement depend on the target environment. Refer to "[Linking Programs](#)" on page 6-18 for more information.
5. Include any additional linkage editor statements you might require.

Linking API Stub Programs to Run in a Shell

When linking an API Stub program to run in a POSIX environment, specify the following object file to be linked with your program:

```
$ORACLE_HOME/lib/orastbl.o
```

If there are unresolved external references for symbols whose names begin with SQL, then ensure orastbl.o is included in the linkedit. This API stub resolves any calls in the generated code. If orastbl.o is correctly included, then the problem is probably caused by a missing entry point to the API stub routine. You can contact Oracle Support Services for additional assistance.

Sample Commands for Compiling and Linking API Stub Programs in a Shell

The following are sample commands for compiling and linking API stub programs in a shell:

```
COBOL:
export STEPLIB=SYS1.SIGYCOMP
export COBOPT='QUOTE,DLL,OPTIMIZE,SOURCE,LIST'
```

```

cob2 pgm_name.cbl -v \
$ORACLE_HOME/lib/orastbl.o \
-I. \
-I$ORACLE_HOME/rdbms/public \
-I$ORACLE_HOME/precomp/public \
-o pgm_name 2>pgm_name.err

```

C/C++:

```

c89 -I. \
-I$ORACLE_HOME/rdbms/public \
-I$ORACLE_HOME/precomp/public \
-o pgm_name $ORACLE_HOME/lib/orastbl.o \
pgm_name.c \
2>pgm_name.err

```

Linking DLL Stub Programs in a Shell

When linking a DLL stub program to run in a POSIX environment, specify the object files and side decks listed in [Table 6-4](#) for the compiler and environment.

Table 6-4 Binding with DLL Stubs

Platform	Language	Bound With	XPLINK	Must Specify
OCI Apps (no Pro*C/C++ or OCCI)	ISO C	c89	Not Recommended	\$ORACLE_HOME/lib/orastbp.o \$ORACLE_HOME/lib/LIBCLNTS.x
	Pre-ISO C++	c++	Not Recommended	\$ORACLE_HOME/lib/orastbp.o \$ORACLE_HOME/lib/LIBCLNTS.x
	ISO C++	c++	Recommended	\$ORACLE_HOME/lib/orastbp.o \$ORACLE_HOME/lib/LIBCLNTS.x
OCCI Apps (no Pro*C/C++, or mixed with OCI)	Pre-ISO C++	c++	Required	\$ORACLE_HOME/lib/orastbp.o \$ORACLE_HOME/lib/LIBCLNTS.x \$ORACLE_HOME/lib/LIBOCCI.x
	ISO C++	c++	Required	\$ORACLE_HOME/lib/orastbp.o \$ORACLE_HOME/lib/LIBCLNTS.x \$ORACLE_HOME/lib/LIBOCCI.x
	ISO C++	Binder	Required	\$ORACLE_HOME/lib/orastbz.o \$ORACLE_HOME/lib/LIBCLNTS.x \$ORACLE_HOME/lib/LIBOCCI.x

Note: If an application uses DLL stubs, the DLL stub should be included in the main program. Application DLLs and other subroutines loaded by the main program should not include the DLL stub.

Sample Commands for Compiling and Linking DLL Stub Programs in a Shell

The following are sample commands for compiling and linking DLL stub programs in a shell.

C:

```
c89 -I. \
-I$ORACLE_HOME/rdbms/public \
-o pgm_name $ORACLE_HOME/lib/orastbp.o \
$ORACLE_HOME/lib/LIBCLNTS.x \
-W "0,dll,langlvl(EXTENDED),ENUMSIZE(INT)" \
pgm_name.c \
2>pgm_name.err
```

C++:

```
c++ -I. \
-I$ORACLE_HOME/rdbms/public \
-o pgm_name $ORACLE_HOME/lib/orastbp.o \
$ORACLE_HOME/lib/LIBCLNTS.x \
$ORACLE_HOME/lib/LIBOCCI.x \
-+ -W "0,xplink,langlvl(EXTENDED),ENUMSIZE(INT)" \
-W "1,xplink" \
pgm_name.cpp \
2>pgm_name.err
```

Binding OCCI programs to Run in Batch and TSO Environments

The following sample JCL, based on standard procedure CBCXCB, illustrates the bind step necessary for an OCCI C++ program in batch and TSO environments.

```
//BIND EXEC PGM=IEWL,REGION=20M,COND=(8,LE,COMPILE),
// PARM='NOMAP,LIST=NOIMP,COMPAT=CURR,OPTIONS=SYSOPTB'
//SYSPRINT DD SYSOUT=*
//SYSLIB DD DSN=CEE.SCEEBND2,DISP=SHR - Note 1 -
//SYSLIN DD DSN=*.COMPILE.SYSLIN,DISP=(OLD,DELETE)
// DD DSN=CEE.SCEELIB(CELHSCPP),DISP=SHR - Note 1 -
// DD DSN=CEE.SCEELIB(CELHS003),DISP=SHR - Note 1 -
// DD DSN=CEE.SCEELIB(CELHS001),DISP=SHR - Note 1 -
// DD DSN=CEE.SCEELIB(C128),DISP=SHR - Note 1 -
// DD DSN=CBC.SCLBSID(IOC),DISP=SHR - Note 2 -
// DD DSN=CBC.SCLBSID(IOSTREAM),DISP=SHR - Note 2 -
// DD DSN=CBC.SCLBSID(COMPLEX),DISP=SHR - Note 2 -
// DD DSN=CBC.SCLBSID(COLL),DISP=SHR - Note 2 -
// DD DDNAME=SYSIN
//SYSLMOD DD - Refer to Note 3 -
//SYSOPTB DD *
 AMODE=31,RENT,DYNAM=DLL,CASE=MIXED
//SYSIN DD *
 INCLUDE '$ORACLE_HOME/lib/orastbz.o' - Refer to Note 4 -
 INCLUDE '$ORACLE_HOME/lib/LIBOCCI.x' - Refer to Note 4 -
 INCLUDE '$ORACLE_HOME/lib/LIBCLNTS.x' - Refer to Note 4 -
*/
```

Usage Notes:

1. Use your installation's run-time library data set prefix.
2. Use your installation's standard I/O library prefix.
3. Point to the PDSE member where the program will be stored.
4. Replace \$ORACLE_HOME with the Oracle home directory.

Building an Oracle XDK API Using Program in a Shell

The Oracle XDK API is available for use by z/OS C and C++ applications that run under a POSIX shell. Any main program or application DLL that uses the Oracle XDK API must be compiled with DLL and NOXPLINK, which is the default, statically linked against the Oracle XDK API and dynamically linked against LIBCLNTS. If any piece of the application uses XPLINK, the XPLINK(ON) LE runtime option must be used. For more information, refer to [Section , "Using the XPLINK\(ON\) LE Runtime Option."](#) If the application accesses an Oracle database, it must follow the rules for main program and application DLL links specified in [Table 6-4](#). It must also use the OCI API, which is compatible with NOXPLINK, or isolate OCCI API use from Oracle XDK API. For more information, refer to [Section , "Using DLLs."](#)

The following are sample commands for compiling and linking Oracle XDK API-using programs in a POSIX shell.

```
C:
c89 -I. \
-I$ORACLE_HOME/rdbms/public \
-I$ORACLE_HOME/xdk/include \
-L$ORACLE_HOME/lib \
-o pgm_name -lxml10 \
$ORACLE_HOME/lib/LIBCLNTS.x \
-W "0,dll,langlvl(EXTENDED),ENUMSIZE(INT)" \
pgm_name.c \
2>pgm_name.err

C++:
c++ -I. \
-I$ORACLE_HOME/rdbms/public \
-I$ORACLE_HOME/xdk/include \
-L$ORACLE_HOME/lib \
-o pgm_name -lxml10 \
$ORACLE_HOME/lib/LIBCLNTS.x \
++ -W "0,dll,langlvl(EXTENDED),ENUMSIZE(INT)" \
pgm_name.cpp \
2>pgm_name.err
```

Usage Notes:

1. Use your installation's run-time library data set prefix.
2. Use your installation's standard I/O library prefix.
3. Point to the PDSE member where the program will be stored.
4. Replace \$ORACLE_HOME with the Oracle home directory.

Using Oracle-Supplied Procedures to Build Applications

Oracle has provided sample batch JCL and POSIX `make` files to simplify building Oracle Precompiler, OCI, and OCCI applications. Chances are you will need to modify them for your system. Consult your system administrator to determine if these are installed on your system and whether they already have been modified for your system.

Sample Batch JCL to Build an Oracle Precompiler Program

The SRCLIB data set created at Oracle Database install time contains sample JCL to demonstrate the precompile, compile, link, and execute phases of a precompiler program in a single batch job. The SRCLIB members that contain the sample JCL are listed in the following table:

Table 6–5 SRCLIB Members Containing Sample Batch JCL to Build an Oracle Precompiler Program

Precompiler	SRCLIB Member
Pro*C/C++	PROCCLGO
Pro*COBOL V10	COB2CLGO
Pro*FORTRAN	PFORCLGO
Pro*PL/I	PPLICLGO

Using Make to Build a Precompiler Program

Oracle provides sample make files for each supported language to precompile, compile, and link precompiler programs. They can be found in the following location, where *precompiler* is either *proc* or *procob*:

```
$ORACLE_HOME/precomp/demo/precompiler
```

If your program depends on non-Oracle libraries, you may have to alter the make files to include them:

Table 6–6 Sample Make Files to Build an Oracle Precompiler Program

Precompiler	Make File
Pro*C/C++	demo_proc.mk
Pro*COBOL	demo_procob.mk

To use the make files, enter a command similar to the following in a shell:

```
$ make -f xxxxx.mk your_program_src_file
```

Pro*COBOL Sample Programs

Sample Pro*COBOL programs are located in the `$ORACLE_HOME/precomp/demo/procob` directory. Some of these programs require that you run the SQL scripts in the `$ORACLE_HOME/precomp/demo/sql` directory.

To build one of the sample Pro*COBOL programs, use the `cd` command to set the working directory to `$ORACLE_HOME/precomp/demo/procob` and issue the following make command:

```
$ make -f demo_procob.mk sample1
```

To build all of the sample Pro*COBOL programs, use the `cd` command to set the working directory to `$ORACLE_HOME/precomp/demo/procob` and issue the following make command:

```
$ make -f demo_procob.mk samples
```

Pro*FORTRAN Sample Programs

FORTRAN does not run in a z/OS shell environment. Therefore, the Pro*FORTRAN sample programs that work on z/OS have been moved to SRCLIB. The sample

programs include FORSMP1-FORSMP4 and FORSMP6-FORSMP11. These programs can be compiled using the PFORCLGO sample procedure, which contains notes and input for those sample programs that require it.

Pro*C/C++ Sample Programs

Many of these programs require that you run the SQL scripts in the `$ORACLE_HOME/precomp/demo/sql` directory.

To build one of the sample PRO*C programs, use the `cd` command to set the working directory to `$ORACLE_HOME/precomp/demo/proc` and issue the following `make` command:

```
$ make -f demo_proc.mk sample1
```

To build all of the sample PRO*C programs, use the `cd` command to set the working directory to `$ORACLE_HOME/precomp/demo/proc` and issue the following `make` command:

```
$ make -f demo_proc.mk samples
```

Batch JCL to Build OCI Programs

There is sample JCL in the SRCLIB data set created at Oracle Database install time that contains sample JCL to demonstrate the precompile, compile, link, and execute phases of OCI programs in a single batch job. The SRCLIB member that contains the sample JCL is CDEMCLGO.

Batch JCL to Build OCCI Programs

There is sample JCL in the SRCLIB data set created at Oracle Database install time that contains sample JCL to demonstrate the compile, link and execute phases of an OCCI program in a single batch job. The SRCLIB member that contains the sample JCL is CPDMCXBG.

Using Make to Build OCI and OCCI Programs

Oracle provides a sample make file to compile and link OCI and OCCI programs. It is located in the `$ORACLE_HOME/rdbms/demo/demo_rdbms.mk` directory.

If your program depends on non-Oracle libraries, you may have to alter this make file to include them.

In order to use this make file, enter a command similar to the following in a POSIX shell environment:

```
$ make -f demo_rdbms.mk build EXE=your_program_src_file SRCS="your_program_name.o ..."
```

Sample OCI and OCCI Programs

Sample OCI and OCCI programs are located in the `$ORACLE_HOME/rdbms/demo` directory. Some of these programs require that you run SQL scripts in the `$ORACLE_HOME/rdbms/demo` directory.

To build one of the sample OCI programs, `cd` to the `$ORACLE_HOME/rdbms/demo` directory and run the following `make` command:

```
$ make -f demo_rdbms.mk build EXE=oci02 SRCS=oci02.o
```

To build all of the sample OCI programs, change to the `$ORACLE_HOME/rdbms/demo` directory and issue the following `make` command:

```
$ make -f demo_rdbms.mk samples
```

To build one of the sample OCCI programs, change to the \$ORACLE_HOME/rdbms/demo directory and run the following make command:

```
$ make -f demo_rdbms.mk buildocci EXE=occiblob SRCS=occiblob.cpp
```

To build all of the sample OCCI programs, change to the \$ORACLE_HOME/rdbms/demo directory and run the following make command:

```
$ make -f demo_rdbms.mk occidemos
```

Building a Traditional Load Module with the Alternate API Stub

The nominal Pro*C/C++, Pro*COBOL, and Pro*PL/I application executable is a PM3 or PM4 program object that must reside in a PDSE or HFS file. This section presents methods for producing traditional load modules. The common component of these methods is an alternate, short name API stub called ORASTBS. ORASTBS is shipped as an object deck, suitable for prelinker or binder input.

Note: These methods can also create a PM2 program object that resides in a PDSE or HFS file.

For more information, refer to *MVS Program Management: User's Guide and Reference*

Method 1: Prelink and Link

This method is suitable for the following types of Oracle API programs: Pro*COBOL, Pro*PL/I, and OCI V7 programs. Existing Pro*C/C++ and OCI V8 programs calling API functions using their truncated names, as documented in prior releases, can use this method as well. The object produced by the compile step, along with the alternate API stub ORASTBS, is passed to the prelinker and the resultant object is then passed to the binder, as shown in the following example:

```
//PRELINK EXEC PGM=EDCPRLK,COND=(4,LT)
//STEPLIB DD DISP=SHR,
//          DSN=SYS1.SCEERUN
//SYMSGS DD DISP=SHR,
//          DSN=SYS1.SCEEMSGP(EDCPMSGE)
//SYSLIB DD DISP=SHR,
//          DSN=ORACLE.V10.OBJLIB
//SYSIN DD DISP=(OLD,DELETE),
//          DSN=&&LOADSET
//          DD DDNAME=SYSIN2
//SYSMOD DD DISP=(NEW,PASS),
//          DSN=&&PLKSET,
//          UNIT=VIO,
//          SPACE=(32000,(30,30)),
//          DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)
//SYSOUT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSIN2 DD *
INCLUDE SYSLIB(ORASTBS)
/*
//LINKEDIT EXEC PGM=HEWL,COND=(4,LT),
//          PARM='LET,LIST,MAP,XREF'
//SYSPRINT DD SYSOUT=*
```



```

//SYSLIB DD DISP=SHR,
// DSN=SYS1.SCEELKED
//SYSUT1 DD UNIT=VIO
//SYSLMOD DD DISP=SHR,
// DSN=USER.LOADLIB(SAMPLE1)
//SYSLIN DD DISP=(OLD,DELETE),
// DSN=&&PLKSET
// DD DDNAME=SYSIN
//SYSIN DD DUMMY

```

The steps to precompile and/or compile the program would precede the PRELINK step. Note that in the case of C programs, NOLONGNAME must be specified on the compile step.

Method 2: Precompile and/or Compile with Name Mapping

This method is suitable for the following types of Oracle API programs: Pro*C/C++ and OCI V8. At precompile and/or compile time, a header file is included which maps the longnames to shortnames.

Inclusion of the header file is triggered by the definition of ORA_SNAME. The object produced by the compile step, along with the alternate API stub ORASTBS, is passed to the prelinker and the resultant object is then passed to the linkage editor, as shown in the following example:

```

//PRECOMP EXEC PGM=PROC,
// PARM='++//DD:SYSPARM'
//STEPLIB DD DISP=SHR,
// DSN=ORACLE.V10.COMDLOAD
//ORA$LIB DD DISP=SHR,
// DSN=ORACLE.V10.MESG
//SYSPRINT DD SYSOUT=*,
// DCB=(LRECL=132,BLKSIZE=1320,RECFM=VB)
//SYSOUT DD SYSOUT=*,
// DCB=(LRECL=132,BLKSIZE=1320,RECFM=VB)
//SYSERR DD SYSOUT=*,
// DCB=(LRECL=132,BLKSIZE=1320,RECFM=VB)
//SYSCOD DD UNIT=SYSDA,
// SPACE=(TRK,(10,10))
//SYSCUD DD UNIT=SYSDA,
// SPACE=(TRK,(10,10))
//SYSPUNCH DD DISP=(,PASS),
// DSN=&&PCCOUT,
// UNIT=SYSDA,
// SPACE=(CYL,(2,1)),
// DCB=(RECFM=FB,LRECL=80,BLKSIZE=800)
//SYSUT1 DD UNIT=SYSDA,
// SPACE=(CYL,(5,5))
//SYSUT2 DD UNIT=VIO
//SYSUT3 DD UNIT=VIO
//SYSLIB DD DISP=SHR,
// DSN=USER.MACLIB
// DD DISP=SHR,
// DSN=ORACLE.V10.H
// DD DISP=SHR,
// DSN=SYS1.SCEEH.H
// DD DISP=SHR,
// DSN=SYS1.SCEEH.SYS.H
// DD DISP=SHR,
// DSN=SYS1.SCLBH.H

```

```

//CONFIG DD DUMMY
//SYSIN DD DISP=SHR,
// DSN=USER.PROGRAM.SRC
//ORA@XXXX DD DUMMY
//SYSPARM DD *
  INAME=//DD:SYSIN
  LNAME=//DD:SYSPRINT
  ONAME=//DD:SYSPUNCH
  CONFIG=//DD:CONFIG
  CODE=ANSI_C
  USER=SCOTT/TIGER
  SQLCHECK=FULL
  DEFINE=ORA_SNAME
/*
//COMPILE EXEC PGM=CBCDRVR,COND=(0,LT),
// PARM=( '/SOURCE,NOMAR,NOSEQ,LIST,RENT,DEF(ORA_SNAME=) ' )
//STEPLIB DD DISP=SHR,
// DSN=SYS1.SCEERUN
// DD DISP=SHR,
// DSN=SYS1.SCBCMP
//SYMSGS DD SYSOUT=*
//SYSLIB DD DISP=SHR,
// DSN=ORACLE.V10.H
// DD DISP=SHR,
// DSN=SYS1.SCEEH.H
// DD DISP=SHR,
// DSN=SYS1.SCEEH.SYS.H
// DD DISP=SHR,
// DSN=SYS1.SCLBH.H
//SYSLIN DD DISP=(,PASS),
// DSN=&&LOADSET,
// UNIT=VIO,
// SPACE=(CYL,(3,3)),
// DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)
//SYSPRINT DD SYSOUT=*
//SYSOUT DD SYSOUT=*
//SYSCPRT DD SYSOUT=*
//SYSUT1 DD UNIT=VIO,
// SPACE=(32000,(30,30)),
// DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)
//SYSUT4 DD UNIT=VIO,
// SPACE=(32000,(30,30)),
// DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200)
//SYSUT5 DD UNIT=VIO,
// SPACE=(32000,(30,30)),
// DCB=(RECFM=FB,LRECL=3200,BLKSIZE=12800)
//SYSUT6 DD UNIT=VIO,
// SPACE=(32000,(30,30)),
// DCB=(RECFM=FB,LRECL=3200,BLKSIZE=12800)
//SYSUT7 DD UNIT=VIO,
// SPACE=(32000,(30,30)),
// DCB=(RECFM=FB,LRECL=3200,BLKSIZE=12800)
//SYSUT8 DD UNIT=VIO,
// SPACE=(32000,(30,30)),
// DCB=(RECFM=FB,LRECL=3200,BLKSIZE=12800)
//SYSUT9 DD UNIT=VIO,
// SPACE=(32000,(30,30)),
// DCB=(RECFM=VB,LRECL=137,BLKSIZE=882)
//SYSUT10 DD SYSOUT=*
//SYSUT14 DD UNIT=VIO,

```

```
//          SPACE=(32000,(30,30)),
//          DCB=(RECFM=FB,LRECL=3200,BLKSIZE=12800)
//SYSIN    DD DISP=(OLD,DELETE),
//          DSN=&&PCCOUT
```

The precompile step is only needed when the program is a Pro*C/C++ application. For OCI V8 programs, only the compile step is needed. In either case, the object produced by the compile step, with the alternate API stub ORASTBS, are passed to the prelinker and the resultant object is then passed to the binder, discussed on page 6-26.

Method 3: Link

This method is suitable for Pro*COBOL and Pro*PL/I applications and Pro*C/C++ applications that are written in C and do not use OCI. The object deck produced by the compile step must not contain any long names. C must be compiled with NOLONGNAME. It and the alternate API stub ORASTBS are passed to the binder, as shown in the following example:

```
//LKED EXEC PGM=HEWL,
// PARM='LET,LIST,MAP,XREF'
//SYSPRINT DD SYSOUT=*
//SYSLIB  DD DSN=SYS1.SCEELKED,DISP=SHR
//        DD                - Refer to Note 1 -
//OBJLIB  DD DSN=ORACLE.V10G.OBJLIB,DISP=SHR
//SYSLMOD DD                - Refer to Note 2 -
//TEXT    DD                - Refer to Note 3 -
//SYSUT1  DD UNIT=VIO
//SYSLIN  DD *
INCLUDE TEXT                - Refer to Note 3 -
INCLUDE OBJLIB(ORASTBS)    - Refer to Note 4 -
                           - Refer to Note 5 -

/*
```

Usage Notes:

1. Include additional libraries required by your program and any runtime libraries required by the native language.
2. This DD statement describes the library in which your linkedited Oracle load module is placed.
3. This DD statement points to the object output from your Oracle Precompiler program's native language compiler.
4. The specific contents of the SYSLIN DD statement depend on the target environment. Refer to ["Linking Programs"](#) on page 6-18 for more information.
5. Include any additional linkage editor statements you might require.

Application Runtime Considerations

Runtime considerations for an Oracle Database for z/OS application depend on the execution environment (batch, TSO, or POSIX) and are essentially the same considerations as those for the Oracle tools and utilities on z/OS, with the addition of whatever resources the application itself requires. Refer to [Chapter 2](#) for a description of basic runtime requirements for the Oracle Database for z/OS client. In this chapter, we provide additional processing details that may be helpful in application design and debugging. Refer to [Table 6-7](#) and [Table 6-8](#) for timelines of Oracle applications that use API and DLL stubs.

Oracle Interface Initialization, Processing, and Error Handling

In most cases, the first call of an API stub application to an Oracle interface function causes the API stub to load and initialize LIBCLNNTS, a large program object containing all of the Oracle interface code for both Oracle Precompiler and OCI applications. (A few precompiler runtime calls, such as the SQLADR function, are serviced entirely within the API stub and do not trigger LIBCLNNTS initialization.)

In an API stub application, the API stub loads LIBCLNNTS using the LE DLL mechanism. In a DLL stub application, LE loads LIBCLNNTS (and, for an OCCI application, LIBOCCI) itself before `main()`, if `main()` is compiled with XPLINK, or at the application's first call to an OCI or OCCI API function. In addition to locating or loading the LIBCLNNTS executable code, LE allocates virtual memory for LIBCLNNTS Writeable Static Area (WSA) and for various internal LE data structures.

After successful LE initialization of LIBCLNNTS, the API stub calls an Oracle-specific initialization function within LIBCLNNTS and certain Oracle initializations are done, such as the processing of global and local environment variable files in non-POSIX applications, as described in [Chapter 2](#).

For an OCI or OCCI POSIX DLL stub application, no Oracle-specific initialization of LIBCLNNTS is required. For a native OCCI (non-POSIX) batch or TSO application, initialization code in the `orastbz` stub calls the Oracle-specific initialization function within LIBCLNNTS.

For an API stub application, after successful Oracle initialization, the original application call to an interface function is routed to the appropriate function within LIBCLNNTS. Subsequent interface calls from the application skip the loading and initialization activities and are routed directly to the target LIBCLNNTS function. OCI and OCCI interface calls in DLL stub applications go directly from the application to LIBCLNNTS or LIBOCCI.

For an API stub application, errors that arise during LIBCLNNTS loading and initialization may be detected by LE or by Oracle code. In most error situations control ends up back in the API stub, which issues messages describing the error and then issues a User 2010 (U2010) ABEND with an accompanying reason code. With one exception, these descriptive messages are issued to the LE message file using the LE CEEMOUT function. The exception is when the API stub detects that the caller is not LE-enabled. In this case, CEEMOUT is not available and the descriptive message is written to the job log with WTO. For more information about error messages, refer to the *Oracle Database Messages Guide for IBM z/OS on System z*.

On all Oracle interface calls from an API stub application, both initial and subsequent, the API stub establishes an LE condition handler for the duration of the call. This means that severe errors detected or raised by LE during either initialization or Oracle call processing cause control to pass to the Oracle handler, which issues message MIC012E to indicate what phase of Oracle API processing was occurring and to display the LE condition token associated with the error. The Oracle handler returns control (percolates) to LE, which passes control to any condition handlers established before Oracle's, including those that may have been established in the application. If there are no other handlers, or the ones there also percolate the error, control ultimately reaches LE's default handler which terminates the enclave.

Normal Oracle processing errors (such as failure to connect to a target server) do not raise LE conditions and so do not participate in the processing just described. Such errors return various error codes to an application as described in the Oracle Precompiler and OCI documentation. An ABEND during an Oracle interface call normally gets trapped by LE and invokes the condition handling mechanisms. This is true for both User (Uxxxx) and System (Sxxxx) ABENDs unless the application has

disabled LE's ABEND interception with the runtime option TRAP(OFF). Be aware that the U2010 ABEND issued for failed LIBCLNTS initialization is subject to this trapping and condition handling process.

For COBOL applications built with the DYNAM option, the following exceptions apply:

- Since the COBOL main module does not include the API stub, the initialization normally performed by the API stub is deferred until the first SQL function is loaded into memory and then called.

To load the first SQL function the system must load the module ORASTBX. ORASTBX resides in the CMDLOAD data set, which also contain LIBCLNTS. If there is a problem loading LIBCLNTS, the API stub initialization will indicate so with a messages similar to the following, followed by a U2010 abend:

```
MIC0022E Failed to load Oracle API module LIBCLNTS - errno 205,
EDC5205S DLL module not found.
MIC011E Oracle API processing error, reason code 002
```

If there is a problem finding or loading ORASTBX, an error will be reported by the COBOL runtime and an S806 abend may result.

- Because the COBOL main module does not include the API stub, there is an issue with LE initialization. The COBOL main module must be marked, so that when LE initialization is performed for the COBOL components, some LE initialization is done to prepare for calling the C components within ORASTBX. This is done by including ORADMYC during the linking of the COBOL module. If the C part of the LE initialization is not perform, then the following error will result:

```
MIC011E Oracle API processing error, reason code 014
```

This is followed by a U2010 abend with reason code 14. Reason code 14 uniquely identifies this as the build-time mistake: ORADMYC was not included.

- Because of the missing API stub, the CEEBXITA routine is absent from the enclave, which excludes a COBOL DYNAM module from participating in nested enclave situations.

Refer to Tables [Table 6-7](#) and [Table 6-8](#) for an overview of the timeline for API stub and DLL stub applications.

Table 6-7 Timeline for Oracle Applications Using API Stubs

When	Action
At bind or link-edit time	<ul style="list-style-type: none"> ■ The API stub, with routines for each API, is included ■ Oracle API references are bound to API stub routines
During LE initialization, in the stub's CEEBXITA exit	Establish support for multiple LE enclaves
At first Oracle API call, its API stub routine	<ul style="list-style-type: none"> ■ Explicitly load stub (COBOL DYNAM only) ■ Explicitly load LIBCLNTS DLL ■ Call LIBCLNTS DLL to initialize ORA\$ENV (non-POSIX only)
At each Oracle API call, its API stub routine	<ul style="list-style-type: none"> ■ Establish recovery routine for API calls ■ Call API routine in LIBCLNTS ■ Remove recovery routine for API calls

Table 6–7 (Cont.) Timeline for Oracle Applications Using API Stubs

When	Action
At normal Termination, the stub's CEEBXITA exit	Clean up multiple LE enclave support

Table 6–8 Timeline for Oracle Applications Using DLL Stubs

When	Action
At bind time	<ul style="list-style-type: none"> ■ The DLL stub is included (main programs only) ■ OCI API references are resolved by the LIBCLNTS side deck ■ OCCI API references are resolved by the LIBOCCI side deck
During LE initialization, the stub's CEEBXITA exit	Establish support for multiple LE enclaves
Before main() (if app uses XPLINK) or before first API call (otherwise)	<ul style="list-style-type: none"> ■ LE loads LIBCLNTS DLL ■ LE loads LIBOCCI DLL (OCCI only) ■ Call LIBCLNTS DLL to initialize ORA\$ENV (non-POSIX only)
At each Oracle API call	Call API routine in LIBCLNTS (OCI) or LIBOCCI (OCCI)
At normal Termination, the stub's CEEBXITA exit	Clean up multiple LE enclave support

See Also: IBM language-specific documentation and to the *IBM Language Environment Programming Guide* for details on LE condition handling, condition tokens, the LE message file, and related topics

Application Resources and Cleanup

The freeing of certain Oracle client interface resources (including allocated virtual memory, open files or data sets, and dynamically-loaded program code) may not occur until the application's LE enclave terminates. In some application designs this can be long after the application's last Oracle interaction, giving the appearance of a memory leak or similar resource issue. This is expected behavior and not a defect. If you want to ensure that all Oracle-related resources allocated by a client are freed, either the associated LE enclave or the associated z/OS task (TCB) must terminate.

The foregoing point refers only to resources in the client environment, not to resources allocated on the client's behalf in a target Oracle database server. Freeing of resources in the server occurs when the server recognizes that the client is disconnected.

Normally this happens when the client logs off of the server through the appropriate OCI or precompiler mechanism. If the application fails to log off, the session persists in the server and server resources remain allocated until the z/OS task (TCB) or address space associated with the client terminates, either normally or abnormally. If the client is connected to the server through TCP/IP protocol, there may be a discernable delay before the server recognizes that the client has terminated.

If a z/OS client application uses cross-memory (XM) protocol to connect to a local Oracle database server and the associated z/OS task terminates either normally or abnormally without logging off, a z/OS resource cleanup routine executes automatically to inform the server that the client is gone. Message MIS0215I is issued

by the resource cleanup routine, reporting the status of the call to inform the server. This message is issued to the log, through WTO, as LE services are terminated at this point. Appearance of this message in an application suggests that proper logoff operations were not done.

Developing Oracle Applications for the CICS TS Environment

Oracle Access Manager for CICS TS provides the interface between Oracle Precompiler programs (COBOL, C languages, or PL/I) and the Oracle database through a CICS TS transaction. CICS TS programs that access the Oracle database are built by precompiling the program containing Oracle SQL statements with the appropriate Oracle Precompiler, translating the output produced with the CICS TS translator or the integrated translator provided with IBM Enterprise COBOL, and compiling and linking the program with the Oracle ORACSTUB object built as part of the Access Manager for CICS TS configuration. ORACSTUB contains code to resolve calls to Oracle code from the application program and the name of the Oracle Access Manager for CICS TS instance, also known as the adapter name, which processes the requests.

CICS TS programs do not designate a target database. Instead, the program communicates with a target adapter named in ORACSTUB. The adapter communicates with a designated database which is defined in the configuration process. This allows the target database to be changed (between test and production, for example) and the Oracle Access Manager for CICS TS configuration to be changed, without changing the CICS TS program.

CICS TS Application Design Considerations

CICS TS application programs accessing the Oracle database are similar to other precompiler applications containing SQL statements that are passed to the Oracle database as the program executes. The following sections describe special considerations in the CICS TS environment:

- [CONNECT Statements](#)
- [Synchronization of Oracle and CICS TS Updates](#)
- [Cursor Considerations](#)
- [Accessing Multiple Oracle Databases](#)
- [Accessing Oracle Database 10g and DB2 Databases in a Single Transaction](#)
- [Additional SQL Statement Restrictions](#)

CONNECT Statements

Explicit CONNECT statements are supported but result in increased overhead and are not recommended. Authentication can be determined with several methods as described in the *Oracle Database System Administration Guide for IBM z/OS on System z*. If used, note the following:

- The CONNECT...AT clause is not supported
- When using CICS SYNCPOINT, if there is more than one CONNECT statement within the same CICS TS transaction, a CICS syncpoint is required prior to a second, or subsequent CONNECT statement. This applies whether multiple connect statements are within one program or the program containing the CONNECT statement is given control by a CICS LINK, CICS XCTL, 'CALL', or other method within the same CICS TS transaction.

- Using an explicit CONNECT statement with "/" as the user ID is redundant and not supported. If an explicit CONNECT statement is not used, then the thread table authorization definitions are used for the transaction. This is equivalent to using CONNECT with "/" as the user ID, but it does not incur the overhead of an extra call to Access Manager for CICS TS.

Synchronization of Oracle and CICS TS Updates

CICS TS programs can synchronize Oracle database updates with updates to other data stores in the same CICS TS unit of work. Oracle Access Manager participates in CICS SYNCPOINT processing when COMMIT(CICS) is specified as the commit/recovery mechanism during the configuration process. If this option is used, CICS SYNCPOINT/ROLLBACK triggers commit/rollback and any Oracle SQL COMMIT/ROLLBACK statements will result in an 'ORAP' CICS TS transaction abend.

If COMMIT(ORACLE) is specified as the commit/recovery mechanism, then Oracle SQL COMMIT/ROLLBACK statements trigger commit/rollback, and these updates are not coordinated with updates to other CICS TS data stores.

The RELEASE option of the COMMIT WORK and ROLLBACK statements is not supported and should not be used in a CICS TS program.

Cursor Considerations

The following considerations are related to cursors:

- The precompiler options RELEASE_CURSOR=YES and HOLD_CURSOR=NO are required for CICS TS programs.
- Each cursor that is opened explicitly must have an associated CLOSE statement before a CICS LINK, CICS XCTL, 'CALL' or other method of transferring control to another program, or CICS RETURN.

Accessing Multiple Oracle Databases

Each CICS TS program that accesses an Oracle database can communicate with one Oracle Access Manager for CICS TS instance which is determined by the ORACSTUB linked with the program. Each Oracle Access Manager for CICS TS communicates with one Oracle database server. Access to more than one Oracle database server from a single CICS TS region can be accomplished using the following:

- Oracle database links, which allow all updates to an accessed database to be part of a single program.
- Multiple Oracle Access Managers, where access logic for each database is contained in a separate CICS TS program. An Oracle Access Manager for CICS TS is configured for each distinct server and an associated ORACSTUB is built for each instance of Oracle Access Manager for CICS TS. The appropriate ORACSTUB is then linked with the program.

Accessing Oracle Database 10g and DB2 Databases in a Single Transaction

When accessing Oracle and DB2 data in a single transaction, the DB2 and Oracle access logic must be separated into distinct source programs that are precompiled and compiled separately. They are then linked to act as a single transaction program.

Additional SQL Statement Restrictions

Programs used with Access Manager for CICS TS can only use Oracle data manipulation (DML) SQL statements.

CICS TS EDF and Oracle SQL Statements

If Oracle Access Manager for CICS TS is configured with the EDF option, a SQL statement will be displayed when the call is made to the Oracle database. Additionally, CONNECT statements are displayed as CONNECT (no user id will be displayed), and SQL statements are displayed with variable names (values will not replace variable names).

Environment Variables

Environment variables are a part of the configuration of Oracle Access Manager for CICS TS and apply to all CICS TS programs accessing a particular Oracle Access Manager for CICS TS instance. For more information, refer to the *Oracle Database System Administration Guide for IBM z/OS on System z*.

Considerations for Building a CICS TS application

An Oracle CICS TS program is precompiled in the same manner as other Oracle Precompiler programs. Refer to the section "[Precompiling Programs](#)" on page 6-9. The output of the precompiler (the "&&PCCOUT" data set) is passed to the CICS TS translator. The precompile step must precede the CICS TS translation step or the CICS TS translator will issue a warning for each EXEC SQL statement.

When linking a program to run under CICS TS, add the following linkage editor control statement:

```
INCLUDE SYSLIB(ORACSTUB)
```

The SYSLIB DD statement includes the data set with the appropriate ORACSTUB object. References to ORASTBS, ORASTBL, or AMILS should not be included. If there are any unresolved references for symbols whose names start with SQL, then ensure ORACSTUB is included in the linkedit. If ORACSTUB is correctly included, then contact Oracle Support Services for additional assistance.

AMODE must be set to 31. There are no special requirements for RMODE.

Refer to the machine readable samples in the SRCLIB library for sample JCL used to process COBOL or C language programs.

Running CICS TS Programs

Normal CICS TS practices for defining your program to CICS TS are followed for Oracle Precompiler programs. CICS TS programs access a shared copy of the full SQL interface, called the CICS TS adapter, that must be configured and available before your program begins running. If the adapter is not available, then you receive error AEY9 from CICS TS and you must contact your CICS TS system administrator to have this condition corrected.

Developing Oracle Applications for the IMS TM Environment

Oracle Access Manager for IMS TM allows COBOL, C or PL/I programming language programs running in an IMS MPP, IFP, or BMP region to access an Oracle Database 10g server anywhere in your network. IMS TM programs accessing Oracle Database

10g databases use the normal IMS calls for input and output message processing, data access and synchronization as they would if Oracle Database 10g were not in use.

IMS TM Application Design Considerations

IMS TM application programs accessing the Oracle database are similar to other precompiler applications containing SQL statements that are passed to the Oracle database as the program executes. The following sections describe special considerations in the IMS environment:

- [IMS TM Versions Supported](#)
- [CONNECT Not Supported](#)
- [Synchronization of Oracle and IMS TM Updates](#)
- [Cursor Considerations](#)
- [Accessing Multiple Oracle Databases](#)
- [Additional SQL Statement Restrictions](#)
- [Accessing Oracle Database 10g and DB2 Databases in a Single Transaction](#)
- [Processing of Oracle Database 10g Errors by Your IMS TM Program](#)

IMS TM Versions Supported

The following IMS TM versions are supported by Oracle Database for z/OS:

IMS TM V6

IMS TM V7

IMS TM V8

CONNECT Not Supported

A typical Oracle Database 10g application uses the CONNECT statement to specify which server is to be accessed and to send the server an Oracle user id and password for authentication. With Oracle Access Manager for IMS TM, this information is configured outside of the program:

- The application program does not specify an Oracle server but accesses a specified instance of Access Manager for IMS TM which is configured to communicate with a specific Oracle server. A language interface token(LIT), built as part of the configuration process, is linked with the application program to designate which instance of Oracle Access Manager for IMS TM will be accessed by the application program.
- The Oracle user ID and authentication to be used for IMS TM transactions are related to the IMS PSB name and are defined in the Resource Translation Table (RTT) as part of the configuration process.

See Also: For more information, refer to the *Oracle Database System Administration Guide for IBM z/OS on System z*

Synchronization of Oracle and IMS TM Updates

A typical Oracle Database 10g application might use COMMIT or ROLLBACK statements to control whether database updates are committed or removed. With Oracle Access Manager for IMS TM, these SQL statements are not available. Instead, programs must use native IMS functions (such as GU, SYNC, ROLL, or ROLB) to synchronize both Oracle and non-Oracle updates.

An AM4IMS application program running in a BMP must issue an IMS `CHKP` or `ROLB` prior to termination. For example, a COBOL program would access Oracle, then invoke `CHKP` or `ROLB` before the exiting `GOBACK` statement.

Cursor Considerations

The Oracle Precompiler `MODE` option lets you specify one of several alternatives to normal Oracle SQL processing behavior. This allows applications to adhere more closely to ANSI/ISO rules. These options work under Oracle Access Manager for IMS TM. For example, if `MODE` is set to `ANSI`, then the cursors are closed with each transaction.

Accessing Multiple Oracle Databases

Each program targeted to IMS TM can normally only communicate with one Oracle Access Manager for IMS TM and only one server because it can only be linked with one LIT. Access to more than one Oracle Database 10g database server from a single IMS TM region can be accomplished as follows:

- You can use Oracle database links. With this method, all updates to any of the accessed databases can be part of a single program.
- You can use multiple Oracle Access Managers. To accomplish this, you first design your application so that the access logic for each database is contained in a separate IMS TM program. Then you configure an Oracle Access Manager and a corresponding LIT for each distinct server to be accessed. Finally, you link each of the IMS TM programs with the appropriate LIT.

Additional SQL Statement Restrictions

Use only Oracle DML SQL statements in programs used with Oracle Access Manager for IMS TM.

Accessing Oracle Database 10g and DB2 Databases in a Single Transaction

When accessing Oracle and DB2 data in a single transaction, the DB2 and Oracle access logic must be separated into distinct source programs that are precompiled and compiled separately. They are then linked to act as a single transaction program.

Processing of Oracle Database 10g Errors by Your IMS TM Program

Oracle errors that are considered application-oriented are always returned to the transaction program to be handled by the application logic. These include message `ORA-0001`, all errors in the range of messages `ORA-1400` to `ORA-1489`, and user-defined error messages in the range of `ORA-20xxx`. It is the responsibility of the application developer to include suitable error handling logic for these types of errors.

All other errors are considered system-oriented. These include errors associated with loss of the connection to the target Oracle9i database server and simpler errors such as `ORA-0942`. How Oracle Access Manager for IMS TM handles these errors is determined by an IMS option called the region error option (REO).

The REO can specify that system errors are to be passed to the transaction for handling that is identical to application-oriented errors. Alternatively, the REO can specify that system errors `abend` and `requeue`, or `abend` and `discard` the transaction.

Which REO to use is decided by the application developer and the IMS administrator. The REO is not specified by or in the application program. It is an Oracle Access Manager for IMS TM configuration parameter.

Environment Variables

Environment variables are a part of the configuration of Oracle Access Manager for IMS TM and are defined for IMS transactions in the Resource Translation Table (RTT). For more information, refer to the *Oracle Database System Administration Guide for IBM z/OS on System z*.

Considerations for Building an IMS TM application

This section describes considerations for linking a program to run under IMS TM:

- Add the following linkage editor control statements to the SYSLIN DD statement:

```
INCLUDE SYSLIB(AMILS)
INCLUDE SYSLIB(lit-module)
```

The lit-module statement is the customer-generated language interface token(LIT) described in this chapter. For more information, refer to the *Oracle Database System Administration Guide for IBM z/OS on System z*.

- Do not include references to ORASTBL, ORASTBS, or ORACSTUB.
- If there are unresolved external references for symbols whose names begin with SQL, then ensure AMILS is included in the linkedit. This API stub resolves any calls in the generated code. If AMILS is correctly included, then the problem might be caused by a missing entry point to the stub routine. You can contact Oracle Support Services for additional assistance.

Running IMS TM Programs

Normal IMS TM practices for defining your program to IMS are followed for Oracle Precompiler programs. IMS TM programs access a shared copy of the full SQL interface that must be configured and available before your program begins running. If the interface is not available, then you receive error 3042 from IMS TM and you must contact your IMS system administrator to have this condition corrected.

Migration from Earlier Oracle Versions

If you have existing Oracle-accessing applications on z/OS that were developed with an Oracle Database release before Oracle Database 10g, read this chapter to understand the issues or considerations for those applications. Which issues or considerations apply depend on the applications and on the Oracle Database version with which they have been running.

This chapter contains the following sections:

- [Overview](#)
- [Migrating from Oracle8i](#)
- [Migrating from Oracle9i](#)

Overview

There are two independent aspects in an application migration. One is migration of the z/OS client-side components (the Oracle program interface code, message library, Access Managers, and Oracle tools and utilities). The other is migration of the target Oracle server used by the applications. Typically, these two migrations are not done simultaneously. In general, applications built under an older Oracle version and using runtime libraries from that version will run against Oracle Database 10g without changes. This means the server migration to Oracle Database 10g can be done first and client-side applications can be migrated later, when it is convenient or at the point where you want the application to exploit features that are new in Oracle Database 10g.

This flexibility is somewhat reduced on z/OS (compared to some other Oracle platforms) due to significant improvements in Oracle's integration with z/OS in the previous two versions. Depending on the release you are migrating from and on what facilities your applications use, you may need to rebuild or make external (for example, JCL) changes in certain applications in order to run them with Oracle Database 10g client-side libraries and components. In any case, you should not attempt to migrate client-side components to Oracle Database 10g before the target Oracle servers have been migrated.

Migrating from Oracle8i

Client applications built with and using libraries from Oracle8i OSDI will continue to work without changes when run against Oracle Database 10g. They must continue to use the stubs, CMDLOAD and MESH data sets, or (in the case of POSIX shell applications) the ORACLE_HOME structures from the Oracle8i OSDI distribution. You cannot mix these applications with Oracle9i or Oracle Database 10g applications in a

multi-task or multi-enclave design (where they would share the same STEPLIB and/or ORA\$LIB as Oracle9i or Oracle Database 10g programs). Be aware that all non-POSIX Oracle8i clients in z/OS use the OSDI Network service when accessing a remote Oracle server. You must continue to run an Oracle8i OSDI Network service until all such clients have been migrated.

Migration Checklist

Migrating an Oracle8i client application to use Oracle Database 10g libraries and infrastructure requires the following:

- Support for Oracle SQL*Net V1-style connection strings was removed starting with Oracle9i. If you have applications that rely on such strings, they must be changed to one of the connection specification mechanisms described in [Chapter 3, "Oracle Net and Server Connections on z/OS."](#) SQL*Net V1-style connection strings can be recognized by the full colon following the driver identifier, which was usually a single letter or digit. An example (including the Oracle user ID and password) is as follows:

```
scott/tiger@z:ora3
```

Several different driver identifiers were used on OS/390, including M, F, W, and Z for cross-memory operation and T for TCP/IP. This notation can appear in JCL or scripts (for example, in a tool or utility parameter), in an input file, or in precompiler or OCI program source code.

- Applications that use an Oracle8i Net PROTOCOL=XM address descriptor, either directly, in a tnsnames file, as part of Access Manager configuration, or in a TWO_TASK environment variable, may require changes to the XM address data. Refer to [Chapter 3](#) for details on the PROTOCOL=XM address.
- Net PROTOCOL=IXCF is no longer supported as of Oracle9i. Comparable capability is provided by routing TCP/IP over the cluster interconnect and using PROTOCOL=TCP. Client applications using PROTOCOL=IXCF must be changed to use PROTOCOL=TCP.
- Starting with Oracle9i, only IBM Language Environment-conforming compilers are supported for batch, TSO, and POSIX precompiler and OCI applications. If your application currently is compiled with a non-LE compiler, you must recompile it with an LE compiler currently supported by IBM. Re-precompiling of Pro*C, Pro*COBOL, or Pro*PL/I applications is recommended but not required. Refer to [Chapter 6, "Developing Oracle Applications to Run on z/OS"](#) for details on precompiling and compiling for Oracle Database 10g.
- Starting with Oracle9i, Oracle's own C runtime library is replaced by IBM Language Environment for runtime services. Backward compatibility is provided for some (but not all) Oracle runtime features, particularly filespec notation. The compatibility is mostly optional but it is enabled by default. Refer to the section ["Oracle C Runtime Compatibility"](#) on page 2-29 for additional information. If your application uses an Oracle runtime feature for which backward compatibility is not provided, or if you disable the backward compatibility altogether, your application must be changed to use comparable LE facilities.
- Precompiler and OCI applications from Oracle8i must be relinked or rebound to pick up the Oracle Database 10g linking stub before they can run with the Oracle Database 10g CMDLOAD and MESH data sets. Refer to [Chapter 6](#) for details on the stubs and on linking or binding procedures.

- For applications that run under CICS TS or IMS TM, the Oracle Database 10g version of the corresponding Access Manager must be installed and configured. Contact your CICS or IMS administrator if you are unsure about this.
- For applications that run in TSO or batch, you must change to Oracle Database 10g runtime services and libraries and ensure that IBM LE runtime is available. In most cases this involves changing JOBLIB/STEPLIB and ORA\$LIB DD statements or allocations to refer to the Oracle Database 10g versions of these data sets. Refer to [Chapter 2](#) for details on Oracle client runtime requirements.
- For applications that run in a POSIX shell, ORACLE_HOME should be changed to refer to the Oracle Database 10g ORACLE_HOME directory, and corresponding changes must be made to environment variables such as PATH and LIBPATH.
- Starting with Oracle9i, TSO, Access Managers, and batch clients connecting to remote Oracle servers interact directly with the z/OS TCP/IP protocol implementation instead of going through the OSDI Network service. (POSIX shell clients already do this in Oracle8i.) IBM's TCP/IP implementation requires POSIX "dubbing" of a program that is not already a POSIX process. The z/OS user ID associated with a batch job, CICS TS or IMS TM started task or jobname, or TSO session that connects to a remote Oracle server must be capable of being dubbed. Consult your z/OS system or security administrator if you are unsure about meeting this requirement.
- Be sure to read the considerations covered in the section "[Migrating from Oracle9i](#)" on page 7-3

Migrating from Oracle9i

Applications that were built under Oracle9i will continue to work when run against an Oracle Database 10g server when using the Oracle9i runtime components (such as CMDLOAD and MMSG data sets and ORACLE_HOME). In addition, Oracle9i applications will run unmodified using the Oracle Database 10g runtime components. This simplifies migration and is useful in certain situations where multiple programs (possibly mixed Oracle9i and Oracle Database 10g programs) must share one set of runtime components.

There are, however, some significant changes in the z/OS Oracle client-side implementation in Oracle Database 10g. Most changes are internal improvements with little or no external manifestation, but some are things you may want to exploit and a few are external behavior changes that could impact existing jobs, scripts, or procedures. Read each of the following sections to determine if there are changes that could affect your applications.

Normalized File Access

In Oracle9i, the non-POSIX and POSIX versions of Oracle's client-side programs on z/OS (including tools and utilities as well as the program interface code LIBCLNTS) were merged so that the same program was used in both environments. However, most of the client components that could use either HFS files or z/OS data sets remained predisposed to one or the other type based on the LE POSIX setting: when POSIX was OFF, filespecs were assumed to be data sets; when POSIX was ON, they were assumed to be HFS files.

Starting in Oracle Database 10g, client-side file access is mostly "agnostic" as to the POSIX setting, allowing data sets to be used in a POSIX application and HFS files to be used in non-POSIX applications. The syntax of a filespec indicates which type of file is being used and the POSIX setting is consulted only when a filespec is ambiguous,

meaning not self-identifying as to type. Refer to [Chapter 2](#) for details on filespec syntax and related processing.

Global Environment File

Oracle Database 10g for z/OS introduces a new feature, the global environment file. This feature permits the installation to set specific systemwide defaults for environment variables in Oracle client programs running in TSO or batch (POSIX OFF). The defaults can be overridden using a local environment variable file specified using the ORA\$ENV DD statement. Consult with your system administrator to determine if the global environment file is configured on your system. For additional information refer to the section "[Environment Variables](#)" on page 2-9.

Use of LE Exit CEEBXITA

Starting with Oracle Database 10g, the Oracle linking stubs for TSO, batch, and POSIX applications contain an IBM Language Environment exit, CEEBXITA. When linked into your application this exit allows Oracle code to "clean up" Oracle-specific resources when the application terminates. This design is necessary to allow execution of multiple Oracle-accessing applications (your own applications or Oracle tools or utilities) on a single z/OS task (TCB). With this exit, Oracle supports both nested LE enclaves and serial (successive) execution of Oracle-accessing programs from a single z/OS task. Oracle's use of this exit means that no installation-specific CEEBXITA exit can be used with an Oracle Precompiler or OCI application on z/OS.

A similar version of this exit was provided for Oracle9i in the fix for Oracle bug 3431417. If your Oracle9i application was linked with a version of the stub that includes this fix, the application can participate in a multiple enclave task application on z/OS. Refer to the section "[Application Design Considerations](#)" on page 6-3 for additional details.

Oracle Runtime Compatibility

Starting with Oracle Database 10g, most of the backward compatibility for the Oracle C runtime library is optional, controlled by the ORA_RTL_COMPAT environment variable. This means the backward compatibility can be disabled, causing filespecs that begin with "/dd/" or "/dsn/" to be treated as HFS files (as their syntax indicates).

A few adjustments were made in Oracle Database 10g to the logic that supports backward compatibility for Oracle runtime C standard file redirection operators in non-POSIX batch and TSO environments. Unlike other aspects of Oracle runtime backward compatibility, this feature cannot be disabled with an environment variable.

Refer to "[Oracle C Runtime Compatibility](#)" on page 2-29 for additional details.

SYSOUT Filespec in Clients

In Oracle Database 10g, a filespec providing explicit designation of JES spool output is now usable for most client-side output files. This provides an alternative to DD filespecs for creating spooled output and may be particularly useful in POSIX shell applications (where supplying DD statements is awkward). Refer to the section "[File Types and Filespec Syntax](#)" on page 2-14 for additional details.

SQL*Loader Changes

Significant internal changes were made to the SQL*Loader utility in Oracle Database 10g. Mostly these concern the defaulting and processing of the various files

SQL*Loader reads or writes in non-POSIX environments. Some of the changes cause external behavior differences and may affect existing SQL*Loader batch jobs or TSO procedures. Review the following sections before you attempt to migrate existing SQL*Loader jobs or scripts to use SQL*Loader from Oracle Database 10g.

Interpretation of DDN Keywords

In previous Oracle releases, filespecs supplied with the control file keywords INDDN, BADDN, and DISCARDDN were treated exactly the same as INFILE, BADFILE, and DISCARDFILE respectively. In Oracle Database 10g, the values supplied with INDDN, BADDN, and DISCARDDN are treated as 1-character to 8-character DD names; they cannot specify a data set or HFS file name nor can they include a DD:, //DD:, or /DD/ prefix. Existing loader jobs or scripts that use INDDN, BADDN, or DISCARDDN must be modified for SQL*Loader in Oracle Database 10g. If the associated file is in fact a DD, remove any DD:, //DD:, or /DD/ prefix from the supplied value. If the value is meant to be processed as a data set or HFS file name, change the control file keyword to INFILE, BADFILE, or DISCARDFILE.

Default Filespecs for DD-type Data Files

In previous Oracle releases, when multiple input data files were specified as DD-type files in a single load, if corresponding bad and discard files were not specified they would default to the same DD names (BAD and DISCARD respectively) for every data file. This caused rejected and discarded rows from all data files to be written to the same DDs. In Oracle Database 10g, the defaults for these files (when the corresponding data file is a DD type) are //DD:BAD and //DD:DISCARD for the first data file, //DD:BAD2 and //DD:DISCAR2 for the second, and so on for up to 99 data files.

Default Filespecs for Data Set Name Files

When a loader data file is specified as a z/OS data set name filespec in Oracle Database 10g, defaults for the bad and discard files are constructed using the high-level qualifier (if any) of the control file, the "base" portion (minus any high level qualifier) of the data file data set name, and the extension suffixes BAD and DSC. This matches the default construction of POSIX HFS filenames and reflects the notion that the bad and discard outputs of the load are associated with the load application (its control file) rather than with the input data file. In previous releases, construction of these data set names did not involve the control file high level qualifier.

Default Bad/Discard DCB attributes

In Oracle Database 10g, when the bad and discard files are data sets without explicitly-supplied DCB attributes (either coded on the DD statement or, for an existing data set, already present in the VTOC), DCB attributes are derived automatically based on the attributes of the associated data file. If these files do have specific DCB attributes, SQL*Loader checks them to ensure that the logical record length is the same or greater than that of the data file and fails the load if not. In previous releases, DCB attributes for the bad and discard files were not defaulted or checked based on the attributes of the data file.

Refer to the section "[SQL*Loader](#)" on page 4-8 for z/OS-specific information about Oracle Database 10g SQL*Loader.

Environment Variables Used by Oracle Database for z/OS

This section lists the environment variables used by Oracle components running on z/OS. Not included are variables like `PATH` and `LIBPATH` that are significant to Oracle programs but are processed only by IBM z/OS components.

Listed with each variable are codes indicating the environments where the variable is meaningful. Be aware that each of these environments has distinct mechanisms for specifying environment variables. The environment codes are:

- C: CICS client
- I: IMS client
- N: Native (non-POSIX) batch and TSO client
- P: POSIX client
- S: Oracle server

Environment Variables Used by Oracle Database

The environment variables used by Oracle Database for z/OS are listed in the following table:

Table A-1 *Environment Variables Used by Oracle Database for z/OS*

Name	Environment	Description
<code>_BPXK_MDUMP</code>	N, P	Defined by LE for control of dump processing; if set to any value, Oracle tools/utilities disable LE TRAP processing so that a normal z/OS dump (for example, <code>SYSMDUMP</code>) can be obtained.
<code>NLS_*</code>	C, I, N, P, S	These variables determine character set, language, and other locale settings. They are described in the <i>Oracle Database Globalization Support Guide</i> .
<code>ORA_RTL_COMPAT</code>	N	Set to "OFF" to disable Oracle C runtime compatibility; any other setting (or not set) enables Oracle C runtime compatibility.
<code>ORACLE_HOME</code>	P, S	Indicates Oracle install components location in POSIX HFS.
<code>ORACLE_SID</code>	N, P	Specifies OSDI SID of target server to be accessed with XM protocol.

Table A-1 (Cont.) Environment Variables Used by Oracle Database for z/OS

Name	Environment	Description
SHELL	P	Normally defined by the z/OS UNIX shell, but used by SQL*Plus to determine shell for HOST command.
SQLPATH	P	Used by SQL*Plus to search for SQL files.
TNS_ADMIN	P	Used by Oracle Net to locate certain files in POSIX HFS.
TWO_TASK	N. P	Specifies Oracle Net TNS service name or address for target server.

Symbols

_CEE_ENVFILE mechanism, LE, 2-11

A

Access Manager for CICS TS

- accessing multiple Oracle databases, 6-34
- accessing Oracle and DB2 databases, 6-34
- application design considerations, 6-33
- configured with EDF option, 6-35
- CONNECT statements, 6-33
- considerations for building applications, 6-35
- cursor considerations, 6-34
- developing applications, 6-33
- environment variables, 6-35
- overview, 1-3
- running programs, 6-35
- SQL statement restrictions, 6-35
- synchronization of Oracle updates, 6-34

Access Manager for IMS TM

- accessing multiple Oracle databases, 6-37
- accessing Oracle and DB2 databases, 6-37
- application design considerations, 6-36
- CONNECT not supported, 6-36
- considerations for building applications, 6-38
- cursor considerations, 6-37
- developing applications, 6-35
- environment variables, 6-38
- overview, 1-3
- processing Oracle errors, 6-37
- SQL statement restrictions, 6-37
- synchronization of Oracle updates, 6-36
- versions supported, 6-36

API programs, Oracle

- building, 6-26
- precompile/compile with name mapping, 6-27
- prelink and link method, 6-26

API stub

- alternate (ORASTBS), 6-26
- description, 2-4
- load modules using alternate, 6-26

applications

- building, 6-9
- CICS TS, 6-33
- compiler options, 6-15

- compiling programs, 6-17
 - design considerations, 6-3
 - dynamic linkage techniques, 6-5
 - error handling, 6-30
 - IMS TM, 6-35
 - interface initialization, 6-30
 - linking programs, 6-18
 - migrating, 7-1
 - multiple LE enclaves, 6-7
 - multitasking, 6-7
 - OCI interface to publish/subscribe, 6-8
 - Oracle Precompiler options, 6-10
 - overview, 1-3
 - POSIX threading, 6-8
 - precompiling programs, 6-9
 - processing, 6-30
 - requirements for basic, 6-4
 - requirements for complex, 6-5
 - resources and cleanup, 6-32
 - runtime access requirements, 2-2
 - runtime considerations, 6-29
 - sample JCL to build Oracle Precompiler program, 6-24
 - software and hardware states, 6-8
 - using Oracl-supplied procedures, 6-23
 - z/OS Assembler language, 6-5
- ATTACH macro, z/OS, 6-7
- Attention interrupts, TSO, 2-4

B

bad and discard files

- attributres, 4-12
- SQL*Loader, 4-11, 4-12

batch

- environment, 2-2
- environment variables, 2-10
- linking programs, 6-18
- parameters, 2-6
- redirection operators, 2-20
- running Oracle Precompilers, 6-13

C

- CALL macro, z/OS, 6-6
- C/C++ compiler

- options, 6-16
- overview, 2-1
- character data
 - Oracle Database for z/OS, 5-1
 - partitioned tables, 5-2
 - SQL queries, 5-1
 - SQL statements, 5-2
- CMDLOAD data set
 - description, 2-3
- COBOL compiler
 - options, 6-16
- COBOL DYNAM option
 - exceptions for COBOL applications, 6-31
 - linking programs using, 6-19
 - OCI functions with callbacks, 6-17
 - ORASTBX, 2-4
 - Pro*COBOL DLL option, 6-17
 - Pro*COBOL precompiler support, 6-12
- compilers, IBM
 - options, 6-15
- compiling programs
 - OCCI programs to Run in Batch and TSO
 - Environments, 6-18
 - overview, 6-17
 - sample commands for shell, 6-17
- cross-memory services, z/OS, 1-3

D

- data files
 - bad and discard attributes, 4-12
 - VSAM, 4-12
- data sets
 - DCB attributes, 2-27
 - name filespecs, 2-22
 - name prefixing, 2-30
 - names, 2-14
 - SYSOUT, 2-16
- Datapump Export and Import
 - file processing on server, 5-7
 - interactive mode, 4-19
 - invocation, 4-17
 - log files, 4-18
 - overview, 4-17
 - processing, 4-18, 4-19
 - return codes, 4-19
- DCB attributes
 - data set, 2-27
 - default, 2-27
 - FNA, 2-27
 - without FNA, 2-28
- DD names
 - description, 2-15
- directory object, Oracle Database for z/OS, 5-4
- dynamic linkage techniques, 6-5

E

- enclaves
 - multiple, 6-7

- Enterprise Manager, 1-4
- ENVARN runtime option, 2-10
- environment variables
 - Access Manager for CICS TS, 6-35, 6-38
 - batch and TSO, 2-10
 - description, 2-11
 - ORA\$ENV file, 2-12
 - overview, 2-9
 - used by Oracle Database for z/OS, A-1
 - z/OS UNIX shell, 2-10
- execution environments, z/OS, 2-2
- Export and Import utilities
 - cross-system capabilities, 4-16
 - examples, 4-16
 - export file, 4-15
 - invocation, 4-14
 - overview, 4-14
 - return codes, 4-16
 - unsupported features, 4-16
- extension processing of file specs, 2-23
- external LOB files (BFILES), 5-6
- external tables, 5-6

F

- FATTR string, 2-25
- fetch() function, C library, 6-6
- file attribute compatibility
 - Oracle C, 2-30
- file names
 - description, 2-13
- file processing
 - Oracle Database for z/OS, 5-3
- files
 - description of Oracle database, 2-13
 - processing, 2-13
 - redirection, 2-19
 - SQL, 4-3
 - standard, 2-19
 - syntax, 2-14
- filespecs
 - data set name parts, 2-22
 - description, 2-13
 - extension procession, 2-23
 - HFS file name parts, 2-22
 - manipulation, 2-21
 - Oracle C runtime compatibility, 2-29
 - runtime compatibility, 2-29
 - syntax, 2-14
- FNA (File Name Augmentation)
 - attribute assignment, 2-25
 - description, 2-24
 - FATTR attrs string, 2-25
 - keywords and descriptions, 2-25
 - syntax transformation (FNAME), 2-24
- FNAME syntax transformation, 2-24

G

- Gateways, Oracle Transparent, 1-2

H

- HFS
 - file name parts, 2-22
- HFS (Hierarchical File System)
 - shell environment, 2-2
- HFS files
 - syntax, 2-18

I

- INTEGER values
 - Pro*COBOL precompiler, 6-12
- interrupt processing
 - TSO and z/OS UNIX shell, 2-28

J

- Java, Oracle Database, 5-6

L

- Language Environment (LE), IBM
 - multiple enclaves, 6-7
 - overview, 1-1
 - runtime, 2-3
- ldap.ora file
 - description, 3-8
- LIBCLNTS program
 - description, 2-4
 - initialization, 6-30
- LINK macro, z/OS, 6-6
- linking programs
 - batch and TSO, 6-18
 - COBOL using DYNAM, 6-19
 - overview, 6-18
 - to run in a shell, 6-20
- linklist library
 - description, 2-3
- listener, Oracle Net, 1-3
- LOAD macro, z/OS, 6-6
- load modules
 - building traditional, 6-26
- LOG_DIRECTORY_CLIENT parameter, 3-7
- LOG_FILE_CLIENT parameter, 3-7
- LPA library
 - description, 2-3

M

- Make
 - building OCI programs, 6-25
 - building precompiler programs, 6-24
- MESG data set
 - description, 2-4
- migrating applications
 - from Oracle8i, 7-1
 - from Oracle9i, 7-3
 - overview, 7-1
- multitasking applications, 6-7

N

- networking
 - Oracle Net, 1-3
 - overview, 1-2
- NLS (National Language Support)
 - description, 2-4

O

- ONAME parameter
 - Pro*PL/I precompiler, 6-13
- ORA\$ENV DD statement, 2-10
- ORA\$ENV environment variable file, 2-11
- ORA\$LIB DD statement, 2-5
- Oracle C runtime
 - compatibility, 2-29
 - data set name prefixing, 2-30
 - file attribute compatibility, 2-30
 - filespec compatibility, 2-29
 - runtime compatibility, 2-29
 - standard files and redirection compatibility, 2-31
- Oracle Call Interface (OC CI)
 - sample JCL to build a program, 6-25
- Oracle Call Interface (OCI)
 - ENUMSIZE(INT), 6-13
 - interface to publish/subscribe, 6-8
 - OTT utility, 6-13
 - overview, 1-4
 - sample JCL to build a program, 6-25
 - sample programs, 6-25
 - using Make to build programs, 6-25
- Oracle Database for z/OS
 - character data, 5-1
 - directory object, 5-4
 - environment variables, A-1
 - file processing, 5-3
 - migrating applications, 7-1
 - security considerations, 5-3
 - SQL queries, 5-1
- Oracle home
 - description, 2-3
- Oracle Net
 - description, 1-3
 - files, 3-6
 - ldap.ora, 3-8
 - listener, 1-3
 - output files, 3-8
 - overview, 3-1
 - protocol and address determination, 3-2
 - protocols, 3-1
 - sqlnet.ora, 3-6
 - TCP protocol, 3-5
 - TCP/IP protocol, 1-3
 - tnsnames.ora, 3-8
 - XM protocol address, 3-4
 - XM protocol description, 3-3
 - z/OS cross-memory services, 1-3
- Oracle Transparent Gateways, 1-2
- ORASTAX runtime module
 - TSO environments, 2-4

ORASTBX module, 2-4
OSDI (Operating System Dependent Interface)
 subsystem, 1-1

P

PA1key interrupts, TSO, 2-4
parameters
 batch (JCL), 2-6
 containing spaces, 2-9
 tool and utility conventions, 2-6
 TSO, 2-7
 where specified, 2-6
 z/OS UNIX shell, 2-8
PATH environment variable
 description, 2-3
PL/I compiler
 options, 6-17
POSIX
 environment, 2-2
 threading applications, 6-8
Precompilers, Oracle
 batch and TSO considerations, 6-13
 commands for running in a shell, 6-15
 configuration files, 6-11
 executables, 6-9
 INCLUDE files, Oracle Precompiler, 6-10
 language-specific coding considerations, 6-12
 options, 6-10
 return codes, 6-12
 sample JCL for Pro*C in batch, 6-13
 sample JCL for Pro*COBOL in batch, 6-14
 sample JCL for Pro*FORTRAN in batch, 6-14
 sample JCL for Pro*PL/I in batch, 6-15
 sample JCL to build programs, 6-24
 using Make to build programs, 6-24
precompiling programs, 6-9
prefixing of data set names, 2-30
Pro*C precompiler
 ENUMSIZE(INT), 6-13
 OTT utility, 6-13
 sample programs, 6-25
Pro*COBOL precompiler
 DYNAM option, 6-12
 RETURN-CODEspecial register, 6-12
 sample programs, 6-24
Pro*FORTRAN precompiler
 sample programs, 6-24
Pro*PL/I precompiler
 ONAME parameter, 6-13
procedures, Oracle-supplied, 6-23
program
 runtime access requirements, 2-2
program interface code, Oracle (LIBCLNTS), 2-4
putenv() statement, 2-10

R

Recovery Manager, 1-4
redirection of standard files, 2-19

redirection operators, TSO and batch, 2-20
return codes
 Datapump Export and Import, 4-19
 Export and Import Utilities, 4-16
 Oracle Precompilers, 6-12
 SQL*Loader, 4-13
 SQL*Plus, 4-6
 TKPROF utility, 4-21
RETURN-CODEspecial register
 Pro*COBOL precompiler, 6-12
runtime compatibility
 controlling, 2-29
runtime considerations
 applications, 6-29
runtime program
 access requirements, 2-2

S

security considerations
 Oracle Database for z/OS, 5-3
shell environments
 sample commands for compiling and linking
 programs, 6-20
 z/OS UNIX, 2-2
SQL queries, Oracle Database for z/OS, 5-1
SQL statements
 character data, 5-2
SQL*Loader
 bad and discard files, 4-11, 4-12
 control file, 4-9
 data files, 4-10
 examples, 4-13
 files and filespecs, 4-9
 invocation, 4-8
 log file, 4-10
 overview, 4-8
 return codes, 4-13
SQL*Plus
 EDIT command, 4-3
 examples, 4-7
 files, 4-3
 HOST command, 4-4
 invocation, 4-1
 output interruption, 4-2
 overview, 4-1
 profile names, 4-2
 return codes, 4-6
 SPOOL command, 4-4
 timing processing, 4-6
 unsupported features, 4-6
sqlnet.ora file
 description, 3-6
standard files, 2-19
SYSOUT data sets, 2-16
system() function, C library, 6-6

T

TCP protocol, 3-5

- TCP/IP protocol, 1-3
- TKPROF utility
 - example, 4-21
 - input trace file, 4-20
 - invocation, 4-19
 - output files, 4-20
 - overview, 4-19
 - return codes, 4-21
- tnsnames.ora file
 - description, 3-8
- tools
 - overview, 1-2
- TRACE_DIRECTORY_CLIENT parameter, 3-7
- TRACE_FILE_CLIENT parameter, 3-7
- TSO
 - CALL command, 2-7
 - environment, 2-2
 - environment variables, 2-10
 - filespec, 2-17
 - interrupt processing, 2-28
 - linking programs, 6-18
 - parameters, 2-7
 - redirection operators, 2-20
 - running Oracle Precompilers, 6-13

U

- UNIX System Services, z/OS
 - environment, 2-2
 - environment variables, 2-10
 - interrupt processing, 2-28
 - parameters, 2-8
- utilities
 - overview, 1-2
- UTL_FILE PL/SQL package, 5-4

V

- VSAM data files, 4-12

W

- Workload Manager (WLM)
 - overview, 1-1

X

- XM protocol
 - examples, 3-4
 - Oracle Net, 3-3

Z

- z/OS Assembler language
 - applications, 6-5

