Oracle9*i* Enterprise Edition

User's Guide

Release 2 (9.2.0.1.0) for OS/390

May 2002

Part No. A97312-01

ORACLE

Oracle9*i* Enterprise Edition User's Guide Release 2 (9.2.0.1.0) for OS/390

Part No. A97312-01

Primary Author: Enterprise Platforms Division

Contents

2 Using the OS/390 Database Instance

3 Oracle9*i* Utilities and OS/390 Files

4 Accessing Oracle9*i* Under USS

5 Export and Import Utilities

6 SQL*Loader

7 Oracle Precompilers

8 Oracle Call Interface

9 SQL*Plus

10 Oracle Net

Send Us Your Comments

Oracle9*i* Enterprise Edition User's Guide Release 2 (9.2.0.1.0) for OS/390

Part No. A97312-01

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

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

If you find any errors or have any other suggestions for improvement, please indicate the title and part number of the documentation and the chapter, section, and page number (if available). You can send comments to us at the following e-mail address.:

infoibm_us@oracle.com

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

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

Preface

Read this guide if you are responsible for performing tasks such as:

- Running Oracle tools or utilities, such as SQL*Plus, Export, or SQL*Loader on OS/390 or z/OS
- Designing or developing Oracle applications, using Pro*C, Pro*COBOL, Pro*PL/I, or Oracle Call Interface (OCI), that will run on an OS/390 or z/OS system.
- Running Oracle application programs on OS/390 or z/OS

You need to understand the fundamentals of OS/390 operating systems before using this guide. This guide provides information only on Oracle9*i* products and their interactions with OS/390.

Structure

This guide contains the following chapters and appendixes:

Chapter 1	provides an overview of Oracle9*i* products.
Chapter 2	discusses basic information about installation tools and utilities needed for using Oracle9*i* Enterprise Edition for OS/390.
Chapter 3	discusses the interaction between OS/390 files and Oracle tools and utilities.
Chapter 4	discusses running Oracle tools under OS/390 UNIX System Services.
Chapter 5	describes the functions of the Export and Import utilities as they relate to the OS/390 operating system.

Chapter 6	discusses how the SQL*Loader utility capabilities are used with OS/390.
Chapter 7	discusses OS/390-specific information about using the Oracle Precompilers.
Chapter 8	discusses OS/390-specific information about using the Oracle Call Interface.
Chapter 9	discusses accessing and running SQL*Plus commands and procedures in an OS/390 environment.
Chapter 10	discusses using Oracle Net in an OS/390 environment.
Chapter 11	This chapter describes migration issues from an Oracle8 database with OSDI to an Oracle9*i* database, and with an earlier Oracle8 database with MPM to an Oracle9*i* database.
Appendix A	This appendix describes API short name support.

Related Documents

There are two parts to the Oracle documentation set: OS/390-specific documentation and product-specific documentation. Your site automatically receives both for the Oracle products you have purchased. Use the product-specific documentation to learn how to use a product, and use the OS/390-specific documentation to learn about special requirements or restrictions for using that product under OS/390.

OS/390-Specific Documentation

The OS/390-specific documentation set is used to install, maintain, and use Oracle9*i* for OS/390 products, and consists of:

- *Oracle9i Enterprise Edition Installation Guide for OS/390*
- *Oracle9i Enterprise Edition System Administration Guide for OS/390*
- *Oracle9i Enterprise Edition User's Guide for OS/390*
- *Oracle9i Enterprise Edition Messages Guide for OS/390*

Product-Specific Documentation

Product-specific documentation describes how to use the Oracle9*i* products. The information in these books is constant for all operating systems under which the

products run. Refer to the environment-specific (in this case, the OS/390-specific) documentation for information that does not apply to every environment.

Refer to the *Oracle Technical Publications Catalog and Price Guide* for a complete list of documentation provided for Oracle9*i* products.

Product Name

The complete name for this product is Oracle9*i* Enterprise Edition for OS/390. To maintain readability and conciseness in this document, the Oracle9*i* Enterprise Edition is referred to as Oracle9*i* for OS/390.

In all cases, the referenced product remains Oracle9*i* Enterprise Edition for OS/390.

Tool and Utility Prompts

Certain Oracle tools and utilities operate interactively, displaying a prompt message in TSO or USS sessions when input is expected from the user. For example SQL*Plus uses "SQL>" and Recovery Manager uses "RMAN>". These prompts appear in examples but are not part of the text that is typed by the user.

Storage Measurements

Storage measurements use the following abbreviations:

- K, for kilobyte, which equals 1,024 bytes
- M, for megabyte, which equals 1,048,576 bytes
- G, for gigabyte, which equals 1,073,741,824 bytes

Conventions

Examples of input and output to the system are shown in a special font:

```
//SYSIN   DSN=oran.orav.INSTJCL(member)
```

All output is shown as it actually appears. For input, the following conventions apply:

Convention	Meaning
italic font	indicates that a word or phrase of your choice must be substituted for the term in *italic font*, such as the actual member name. For example: *member*
oran.orav	is the standard example for high-level and second-level data set name qualifiers. Substitute your system's actual high-level and second-level qualifiers. These qualifiers may appear in lowercase or in UPPERCASE typeface.
<> Angle brackets	indicate that the enclosed arguments are required and at least one of the arguments must be entered. Do not enter the brackets themselves.
[] Square brackets	indicate that the enclosed arguments are optional. Do not enter the brackets themselves.
{} Braces	indicate that one of the enclosed arguments is required. Do not enter the braces themselves.
| Vertical lines	separate choices.
. . . Ellipses	indicate that the preceding item can be repeated. You can enter an arbitrary number of similar items.
Other punctuation	must be entered as shown unless otherwise specified. For example, commas and quotes.

Commands, reserved words, and keywords appear in uppercase in both examples and text. A fileid can appear with both uppercase and lowercase text. When portions of a fileid appear in *italics*, the use of *italic characters* indicates that those portions can vary. Reserved words and keywords must always be entered as is, because they have reserved meanings within Oracle.

Documentation Accessibility

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

information, visit the Oracle Accessibility Program web site at **http://www.oracle.com/accessibility/**.

Documents Referenced in this Guide

Oracle9*i* for OS/390 Books:

Oracle9i Enterprise Edition Installation Guide for OS/390

Oracle9i Enterprise Edition Messages Guide for OS/390

Oracle9i Enterprise Edition System Administration Guide for OS/390

Other Oracle Books:

OCI Programmer's Guide

Oracle Intelligent Agent User's Guide

Oracle Label Security Administrator's Guide

Oracle9i Application Developer's Guide - Fundamentals

Oracle9i Database Globalization Support Guide

Oracle9i Database Utilities

Oracle9i JDBC Developer's Guide and Reference

Oracle9i Net Services Administrator's Doc

PL/SQL User's Guide and Reference

*Pro*C/C++ Precompiler Programmer's Guide*

*Pro*COBOL Precompiler Programmer's Guide*

*Pro*PL/I Supplement to the Oracle Precompilers Guide*

Programmer's Guide to the Oracle Precompilers

*SQL*Plus User's Guide and Reference*

IBM Books:

OS/390 Run Time Library Reference

1

Overview of Oracle9*i* Products

Oracle9*i* Enterprise Edition is a high-performance, secure, high-availability database management system that can be deployed across multiple operating systems and hardware platforms, including OS/390. This architecture, unique to Oracle, offers customers a database server with a consistent code base on a variety of platforms, from desktop to UNIX to mainframe. In essence, this frees customers from viewing the hardware and operating system as the platform and in turn view Oracle® Enterprise Edition as the platform. Oracle software on UNIX is the same product as Oracle software on OS/390.

In addition to integrating Oracle9*i* into a distributed environment, Oracle for OS/390 supports traditional transaction processing such as CICS and IMS_TM workloads, as well as Internet computing, where client applications can access Oracle data on OS/390 from any number of distributed sources. This allows customers to exploit the security, reliability, and availability of the mainframe where it counts most, for the data, and use a less premium, commodity environment for Web server applications. Oracle9*i* Enterprise Edition for OS/390 gives customers that flexibility.

The complete name for this product is Oracle9*i* Enterprise Edition for OS/390. This product runs on IBM's z/OS as well as on OS/390. This document uses the term OS/390 to indicate both operating systems. To maintain readability and conciseness in this document, the Oracle9*i* Enterprise Edition is referred to as Oracle9*i* for OS/390. In all cases, the referenced product remains Oracle9*i* Enterprise Edition for OS/390.

The following sections are discussed in this chapter:

- Oracle9i for OS/390 Transparent Gateways on page 1-16

Oracle9*i* for OS/390 Introduction

This release of Oracle9*i* for OS/390 is based on the execution environment known as OSDI which has replaced MPM. OSDI architecture is the S/390-specific execution environment for all Oracle9*i* for OS/390 products. The same Oracle products are supported, and their generic behavior with respect to applications is unchanged. Some of the benefits of using OSDI are: virtual storage constraint relief, enhanced client workload management, improved performance and throughput, and improved reliability, availability, and serviceability. The supported products include the Oracle9*i* Enterprise Edition RDBMS and Oracle Net.

Existing Oracle for OS/390 users should refer to Chapter 11, "Migration Considerations", for further details on migration issues and downward compatibility considerations.

Some of the key features and benefits of Oracle9*i* for OS/390 include the following:

- Real Application Clusters
- Oracle Label Security
- Oracle Spatial
- Oracle Partitioning
- Oracle Advanced Security
- High-Performance Concurrency Control
- Web Integration
- SQL and PL/SQL Languages
- Application Development
- Server-Based Business Rules
- National Language Support
- Data Replication in Distributed Environments
- Advanced Queuing
- Oracle Objects
- Oracle Call Interface
- Oracle for OS/390 Java Support

Real Application Clusters

Oracle9*i* Real Application Clusters is a revolutionary leap beyond Oracle Parallel Server. A separately licensed server option, Real Application Clusters provides your applications with out-of-the-box, near-linear scaling without the need for parallel-aware design changes. Configuring an Oracle instance to use Oracle9*i* Real Application Clusters is simple, and you can easily and quickly add nodes to your cluster as your needs change. Every node can always access all of your data, so if one node needs to be taken off-line, or happens to fail, the remaining nodes will seamlessly pick up and back out any transactions that may need recovery. This kind of continuous availability is required for today's demanding, 24x7 business applications.

On OS/390, this scalability and availability is accomplished without the need for specialized coupling hardware requiring difficult sizing exercises or complex management tasks. Oracle9*i* uses the native, high-speed interconnect available in all supported S/390 and zSeries environments. If coupling hardware is present Oracle9*i* Real Application Clusters will transparently exploit it, but Real Application Clusters does not require coupling hardware—or even a Parallel Sysplex—to bring you the benefits of greater availability and scalability. When combined with the multi-address space capabilities of Oracle9*i*, Real Application Clusters will let you scale your user populations up to the levels required for Internet applications.

Oracle Label Security

Oracle Label Security is but one of many layers of security protection that Oracle9*i* provides to its OS/390 customers. Oracle Label Security builds on the Oracle Virtual Private Database facilities available in Oracle9*i* by attaching access control information directly to your data. With this sophisticated, flexible security option in place in place each row of your data is separately secured. These techniques, used by government and defense organizations to protect sensitive information and provide data separation, can now be applied to enterprises as diverse as application hosting and health care to help meet security requirements in the Internet Age. For example, in application hosting, a subscriber label can be used to separate data among subscribers in the same application. More importantly, your policies are enforced within the database, providing security even if the application is bypassed. Oracle Label Security also includes a sophisticated policy management tool to manage policies, labels, and user label authorizations. Oracle Label Security is a separately licensed option for the Oracle9*i* server.

Note: Oracle Label Security is not supported with this release of Oracle9*i* for OS/390.

Oracle Spatial

If you decide to license the Oracle Spatial server option, it will provide you with an integrated set of functions and procedures that facilitate analysis based on the spatial relationships of associated data. For example, in an Internet-based customer service application Oracle Spatial could help you decide the most cost-effective way to route deliveries and pickups to your customers. Because it's fully integrated with the Oracle9*i* database, Oracle Spatial allows you to seamlessly integrate this spatial data with the rest of your enterprise applications.

Note: Oracle Spatial is not supported with this release of Oracle9*i* for OS/390.

Oracle Partitioning

Oracle Partitioning, a separately licensed option of the Oracle9*i* server, allows you to divide large tables and indexes into pieces that can be separately managed, instead of managing each table as one large object. Partitioning is a technique that provides scalable performance with a large amount of data. Partitioning decreases the time required to perform administrative operations by applying the operations to smaller units of storage, improves performance through increased parallelism, and improves availability by containing the impact of failures.

Database administrators (DBAs) can specify storage attributes for each partition and the placement of the partition within the host file system, increasing the granularity of control for very large databases. Partitions can be individually taken off-line or brought online, backed up, recovered, exported and imported, and loaded, thereby eliminating the time required for management operations. An individual partition can be built for one table partition, bounding the time required for index maintenance operations. Various local and global index strategies are provided. Partition operations can be performed in parallel. Partitions increase availability by containing media and application failures, that is, applications not requiring data from a down or off-line partition continue to run without any impact.

Partitioning is transparent to applications and standard data manipulation language (DML) statements that run against partitioned tables. The Oracle9*i*

optimizer is aware of the partitions; and partitions that do not contain any data for query are eliminated from the search.

Oracle Advanced Security

The separately licensed Oracle Advanced Security server option provides a comprehensive suite of security features to protect an enterprise's networks and to securely extend corporate networks to the Internet. The Oracle Advanced Security option (formerly known as Advanced Networking Option) provides a single source of integration with network encryption and authentication solutions, single sign-on services, and security protocols.

High-Performance Concurrency Control

With other database systems, applications can have contention problems with performance limited by transactions that lock data at the page level or escalate lower-level locks, regardless of available CPU power or I/O bandwidth. Many traditional mainframe shops must have separate windows of time for online workloads and batch workloads to handle this contention. The Oracle9*i* database server uses full, unrestricted row-level locking for data and indexes, and never escalates locks. In addition, Oracle read operations do not prevent concurrent write operations, ensuring maximum concurrent access to data. The Oracle9*i* database server's high-performance, scalable sequence number generator eliminates application contention to obtain unique numeric key values, a common requirement in transaction processing applications. Reverse-key indexes reverse the bytes in index entries, spreading the inserts of consecutive keys among different blocks to eliminate insertion hot spots.

Web Integration

The Oracle9*i* database server can be accessed by the Oracle9*i* Application Server, which can fully integrate your existing Oracle9*i* business applications with Web technology, and safely deploy them inside or outside of your corporate firewall.

The Oracle9*i* Internet Application Server (*i*AS) enables stored procedures to be invoked by clients using Web browsers to generate dynamic Web documents. This means that Web pages are no longer limited to displaying information from static files.

SQL and PL/SQL Languages

The Oracle9*i* database server is 100 percent American National Standards Institute and International Standardization Organization (ANSI/ISO) SQL92 entry-level compliant SQL. This implementation ensures a fully-open application development environment. In addition, the Oracle9*i* database server offers a number of robust SQL extensions that allow complex operations to be expressed in SQL, improving developer productivity by reducing the need for procedural code. Application performance and scalability are enhanced by performing complex data manipulation operations within the Oracle9*i* SQL engine.

PL/SQL, the Oracle procedural extension to SQL, is an advanced fourth-generation programming language (4GL). It offers modern features such as data encapsulation, overloading, collection types, exception handling, and information hiding. PL/SQL also offers seamless SQL access, tight integration with the Oracle database server and tools, portability, and security.

Application Development

Oracle9*i* stored procedures and triggers improve application development scalability and productivity by allowing common procedures to be developed once and maintained in a central location, instead of in every application. Stored procedures and triggers improve application performance and scalability by allowing application logic to be invoked with a single call. This minimizes network traffic and isolates application processing on the server.

Oracle9*i* stored procedure implementation also supports automatic dependency tracking for scalable application development. Oracle9*i* stored procedures can dynamically define and run SQL statements, permitting powerful and flexible procedures. User-defined functions referenced from SQL statements provide productivity by enhancing the power of SQL, and by enhancing performance by running application-specific logic within the server. PL/SQL cursor variables provide a link to a SQL query, allowing stored procedures to encapsulate statically-defined or dynamically-defined queries, and return one or more multirow result sets to the calling application.

The Oracle9*i* database server includes application development features necessary to construct a new generation of sophisticated applications at low cost. Declarative facilities ensure scalable, reliable enforcement of data integrity while minimizing development, maintenance, and administration costs. PL/SQL, an advanced procedural 4GL that is tightly integrated with the Oracle9*i* database server, provides the power to easily express complex business rules as stored procedural code. The Oracle9*i* programmatic interfaces enable third-generation language (3GL) programs

to access and manipulate Oracle9*i* data and schema. The productive development features of the Oracle9*i* database server improve application performance, scalability, and security to support hundreds of applications and thousands of users.

Server-Based Business Rules

The Oracle9*i* database server enforces constraints, either for database integrity purposes or business rules, at the database level, allowing the greatest amount of security and business-rule enforcement. Oracle9*i* achieves this through the use of declarative integrity constraints, database procedures, and database triggers. Deferred constraint checking optionally shifts integrity constraints checking from the end-of-statement operation to the end-of-transaction operation. This simplifies the coding of certain operations involving integrity constraints. Also, many kinds of constraints can be enabled without stopping update activity on the table.

With PL/SQL stored procedures, functions, packages, and database triggers, you can enforce complex business rules at the server level. This improves application performance, scalability, security, and reduces development costs. Procedures and functions can accept arguments from calling client applications, and return one or more result values. Packages group together definitions of related procedures, functions, variables, cursors, and other database objects to improve development productivity.

When rows are inserted, updated, or deleted from the Oracle9*i* tables, database triggers are run automatically, either once per row or once per statement. Database triggers can be used to enforce complex integrity rules within the server. The Oracle9*i* database server trigger implementation is modeled on the draft ANSI and SQL3 specification.

National Language Support

The Oracle9*i* database server supports deployment of heterogeneous Internet and distributed database configurations by automatically and transparently performing any necessary character set conversions. Oracle9*i* National Language Support (NLS) ensures error messages, sort order, date format, and other conventions automatically adapt to the native language. Parameter settings at the Oracle9*i* database server and system levels determine the behavior of individual conventions.

Separate national calendars, including Japanese Imperial, ROC Official, Thai Buddha, Persian, and Arabic Hijrah, are supported. Arabic and Hebrew display

character set support is also available. NLS also supports user-defined characters and calendar years.

Data Replication in Distributed Environments

The benefits of relational technology cannot be achieved without transparent integration of new and existing systems. The Oracle9*i* database server provides distributed database facilities that make the integration of enterprise data practical. Data can be replicated for direct and highly available local access. Data can be accessed remotely using both SQL and PL/SQL procedure calls in a transparent manner, as if the data were local. Also, data can be stored in both Oracle9*i* database servers and non-Oracle servers. No external servers are required, which can complicate management procedures such as recovery.

The Oracle9*i* database offers basic and advanced replication facilities. For information dissemination throughout an enterprise, Oracle9*i* basic replication supports a simple primary site model of replication where one replica is updatable and all others are read-only. For more sophisticated distributed operation systems, including rollover configurations and mass deployment applications such as sales force automation, Oracle9*i* advanced replication supports bidirectional replication and sophisticated conflict detection, resolution, and management tool capabilities that you require.

Both basic and advanced replication are fully integrated into the Oracle9*i* database server. Updates to replicas are captured using internal triggers that run within the server operation for maximum performance.

Advanced Queuing

Oracle9*i* advanced queuing provides direct support in the database for high-performance queuing operations. This capability provides asynchronous operations and eliminates the dependency on external systems for applications requiring high scalability. The enqueue and dequeue operations can shift processing from within a transaction to a background process, improving the transaction response time.

Oracle Objects

Oracle Objects, the Oracle9*i* object-relational paradigm, allows developers to directly define their business objects, such as purchase orders, inventory items, and data warehouse information within the Oracle9*i* database server. This allows

developers of mainstream commercial applications to better manage their business objects.

The Oracle9*i* database server allows you to define custom object types. An object type is typically defined to correspond to some business object, such as a purchase order. The object type might contain multiple fields or attributes, and it might be nested within other object types. More complex objects that include a variable number of items are supported through variable length arrays and nested tables. This allows, for example, a purchase order object type, which can accommodate a variable number of line items.

Oracle9*i* database server development products, including Oracle Call Interface (OCI), Pro*COBOL, and Pro*C, support objects. Client-side caching, complex object retrieval, and navigational access provide client-side processing and minimize the number of network roundtrips between the client and the server. The object type translator utility generates C programming language header files for use in Pro*C and OCI applications.

The Oracle9*i* database server eases the evolution from relational to object-oriented functionality. The Oracle9*i* object-relational capabilities are built on the same solid foundation as the relational functionality.

Objects are completely integrated throughout the Oracle9*i* database server in all levels of the server, and are supported in both SQL and PL/SQL. The strengths of the Oracle9*i* database server are provided with objects, including the sophisticated Oracle9*i* database server concurrency model, industry leading performance, scalability, reliability, manageability, and availability.

Oracle Call Interface

OCI is an application programming interface (API) that allows you to create applications that use the native function calls of the C programming language to access an Oracle9*i* database server and control all phases of SQL statement execution.

Oracle for OS/390 Java Support

Oracle9*i* for OS/390 provides the following support for Java development under UNIX System Services environment.

- Java Database Connectivity (JDBC)
- SQL embedded in Java (SQLJ)
- Oracle9*i* JVM

- XML Support

Java Database Connectivity (JDBC)

Oracle9*i* for OS/390 provides a JDBC thin driver. It is a Type IV driver that is targeted for applet developers. The driver is 100 percent pure Java and complies with the JDBC 1.2.2 standard.

For communicating with the database, the driver includes an equivalent implementation of the Oracle TTC presentation protocol and Oracle Net session protocol in Java. Both of these protocols are lightweight implementation versions of their counterparts on the server. To use this driver, it is not necessary to install any Oracle-specific software on the client.

SQL Embedded in Java (SQLJ)

SQLJ allows application programmers to embed static SQL operations in Java code in a way that is compatible with the Java design philosophy. A SQLJ program is a Java program containing embedded static SQL statements that comply with the ANSI-standard SQLJ Language Reference syntax.

The Oracle SQLJ translator is conceptually similar to other Oracle Precompilers and performs the following checks:

- Checks the syntax of the embedded SQL.
- Checks data types to ensure the data exchanged between Java and SQL has compatible types and proper type conversions.
- Checks SQL constructs against the database schema to ensure consistency.

The SQL methodology of embedding SQL operations directly in Java code is more convenient and concise than the JDBC methodology. In this way, SQLJ reduces development and maintenance costs in Java programs that require database connectivity. When dynamic SQL is required, however, SQLJ supports interoperability with JDBC such that you can intermix SQLJ code and JDBC code in the same source file.

Oracle9*i* JVM

Oracle9*i* for OS/390 provides support for Java (JDK 1.2) from within the Oracle9*i* database.

Java applications can range from the simple standalone application to large, enterprise solutions. This support allows Java stored procedures running inside the Oracle address space to access Oracle databases using the server-side JDBC driver.

The entire JVM operates in the same address space as the database and the SQL engine. This enhances the performance of Java programs and is much faster than executing a remote call to access Oracle.

The server side JDBC driver supports the same features, APIs, and Oracle extensions as the client-side drivers. This makes application partitioning very straightforward. For example, a Java application that is data-intensive, can easily be moved into the Oracle server for better performance, without having to modify the application-specific calls.

XML Support

Oracle9*i* for OS/390 provides support for several XML related tools. They include the XML Parser for Java, the XML Class Generator for Java, and the XML Parser for PL/SQL. They are all implemented in Java.

The XML Parser for Java and the XML Parser for PL/SQL are stand-alone XML components that parse an XML document so that it can be processed by an application.

XML Class Generator for Java creates Java source files from an XML DTD.

Oracle9*i* for OS/390 Utilities

Oracle9*i* for OS/390 contains numerous, easy-to-use utilities for DBAs to use, providing a fully-integrated data management system. Refer to the appropriate Oracle documentation for product-specific functions and use of the Oracle utilities.

The following utilities are discussed in this section:

- Recovery Manager
- Export and Import
- Migration
- SQL*Loader
- Oracle Enterprise Manager
- UNIX System Services

Recovery Manager

The Oracle9*i* Recovery Manager utility uses information about the database to automatically locate, backup, restore, and recover datafiles, control files, and archived redo logs.

Recovery Manager allows database and tablespace point-in-time recovery. It also allows you to simplify normal restore and recover operations. Instead of requiring analysis of logs to perform point-in-time recovery, Oracle9*i* Recovery Manager allows timestamp specification to simplify this procedure.

Export and Import

Export creates an operating system (OS) sequential data set containing data and definitions from the Oracle9*i* database. The data is created in an internal Oracle format. Export helps migrate data from one system to another in conjunction with the companion utility, Import. Export is also used to provide backups of the Oracle tables and database. Several types of Exports are available to you, depending upon authority level within the Oracle9*i* database server.

Import reads the OS-sequential data set created by the Export utility and builds tables and related data in the target Oracle9*i* database.

Migration

This release of Oracle9*i* for OS/390 supports direct upgrade from any Oracle8 release; a migration utility is not required to upgrade from Oracle8 to Oracle9*i*. A migration utility is provided to upgrade from Oracle V7.3.3 to Oracle9*i*.

SQL*Loader

SQL*Loader facilitates the transfer of data from OS/390 data sets into tables in an Oracle9*i* database. A control file is used to describe the data location, data format, and target tables. You can direct data that does not satisfy certain conditions specified in the control file to special files.

When combined with Oracle Net, SQL*Loader allows you to propagate existing data to remote Oracle9*i* database servers. SQL*Loader supports a wide variety of input file formats, including virtual storage access method (VSAM) files and partitioned data sets (PDSs), and accepts DB2 Load Utility/DXT control file syntax.

Oracle Enterprise Manager

With the server-side Oracle Enterprise Manager Intelligent Agent, DBAs can manage the Oracle for OS/390 database from any OEM console. Starting and stopping the database, as well as managing backups with Recovery Manager, can be controlled through the graphical user interface (GUI) on any OEM console.

UNIX System Services

Oracle9*i* includes several tools and utilities designed to run in the OS/390 UNIX System Services (USS) environment. OS/390 UNIX System Services is similar to other UNIX environments, and the Oracle9*i* tools built for OS/390 USS behave like their UNIX counterparts with every few, if any, differences.

Chapter 4, "Accessing Oracle9i Under USS", describes the Oracle OS/390 UNIX System Services environment. This chapter includes information on the Oracle java utilities available under OS/390 USS. Furthermore, additional information on specific USS compatibility is included in the relevant chapters of this guide.

Oracle9*i* for OS/390 Additional Products

A number of additional products are distributed on the Oracle9*i* for OS/390 server distribution tape. They include the following:

- Oracle Access Managers (for CICS and IMS_TM)
- Oracle Precompilers
- Oracle Net
- SQL*Plus

Oracle Access Managers

Oracle Access Managers enable CICS and IMS_TM applications to directly access local OS/390 data or remote Oracle systems. Usage of one of these is included with the Oracle9*i* for OS/390 license.

Oracle Access Manager for CICS

Oracle Access Manager for CICS provides support for CICS transactions to access Oracle data. The Oracle data can be local on the same OS/390 system or remote on any Oracle-supported platform. If the Oracle data is remote, then Oracle Access Manager for CICS routes the transaction's Oracle requests through Oracle Net to the

remote system. The distinction between local and remote Oracle9*i* database server access is transparent to both the transaction program and the user.

CICS transactions written in COBOL or C programming language can be modified to access the Oracle9*i* database server by embedding industry-standard SQL statements in the source code. This does not affect a program's ability to access non-Oracle data stores on the enterprise server. Both the Oracle9*i* database server data and non-Oracle data can be accessed and updated in a single transaction. Oracle Access Manager for CICS provides the appropriate interfaces so that CICS fulfills its normal role as commit, rollback, and recovery coordinator.

Data available on an Oracle platform can be accessed simultaneously by both CICS transactions and the Oracle tools or programs. This provides complete, reliable control over data consistency and integrity. You can begin to leverage the full range of Oracle products and facilities while still utilizing your traditional applications.

Data available on an Oracle platform can be accessed simultaneously by both CICS transactions and other Oracle tools or programs. This provides complete and reliable control over data consistency and integrity. You can begin to leverage the full range of Oracle products and facilities.

Oracle Access Manager for IMS TM

Oracle Access Manager for IMS_TM provides support for IMS_TM transactions to access Oracle data. The Oracle data can be local on the same OS/390 system, or remote on any Oracle-supported platform. If the Oracle data is remote, then Oracle Access Manager for IMS_TM routes the transaction's Oracle requests through Oracle Net to the remote system. The distinction between local and remote Oracle9*i* database server access is transparent to both the transaction program and to the user.

IMS transaction programs written in COBOL or C programming language can be modified to access the Oracle9*i* database server by embedding industry-standard SQL statements in the source code. This does not affect a program's ability to access non-Oracle data stored on the enterprise server. Both Oracle9*i* database server data and non-Oracle data can be accessed and updated in a single transaction. Oracle Access Manager for IMS_TM provides the appropriate interfaces so that IMS_TM fulfills its role as commit, rollback, and recovery coordinator.

Data moved to another platform can be accessed simultaneously by both IMS_TM transactions and other Oracle tools or programs available in the new environment. This provides complete and reliable control over data consistency and integrity. You can leverage the full range of Oracle products and facilities.

Oracle Precompilers

Oracle Precompilers support SQL high-level source programs. The Oracle Precompilers do the following:

- Accept the source program as input.
- Translate the embedded SQL statements into standard Oracle runtime calls.
- Generate a modified source program you can compile, link, and run.

Oracle Precompilers supported on Oracle9*i* for OS/390 include:

- Pro*C
- Pro*COBOL
- Pro*FORTRAN
- Pro*PL/1

Oracle Net

Oracle Net provides network communications between Oracle client applications and the Oracle9*i* database server or Oracle gateways. It operates across different processors and operating systems. Users of one OS/390 system can access:

- An Oracle9*i* for OS/390 instance on the same processor.
- An Oracle9*i* for OS/390 instance on a different processor.
- An Oracle9*i* instance on a non-OS/390 platform.

Oracle Net enables protocol independence. Regardless of the quantity or type of communication protocols used, data access is seamless. Protocols supported for Oracle Net include:

- IBM TCP/IP High Performance Native Sockets (HPNS)
- Oracle Net cross memory driver

SQL*Plus

SQL*Plus enables you to manipulate data using SQL commands. With SQL*Plus you can do the following:

- Enter, edit, store, retrieve, and run SQL commands
- Format, perform calculations on, store, and print query results in report form

- List column definitions for any table
- Access and copy data between databases
- Send messages to and accept responses from a user
- Manage Oracle9*i* database servers

Oracle9*i* for OS/390 Transparent Gateways

There are two Oracle Transparent Gateways available for Oracle9*i* for OS/390:

- Oracle Transparent Gateway for DB2
- Oracle Transparent Gateway for EDA/SQL

Oracle Transparent Gateways are available on separate distribution media. To obtain them, contact your local Oracle representative.

Oracle Transparent Gateway for DB2

Oracle Transparent Gateway for DB2 is tightly integrated with Oracle9*i* for OS/390 and provides Oracle applications with read and write access to DB2 tables. The gateway runs as a started task under OS/390 and uses the DB2 standard Call Attach Facility (CAF) to access tables stored in DB2.

Oracle Net provides network support between the client and the Oracle9*i* database server for any protocol supported on both platforms, and supports TCP/IP between a remote Oracle9*i* database server and the gateway on OS/390. The Oracle9*i* database server can reside on any Oracle-supported platform, including OS/390.

Oracle Transparent Gateway for DB2 is Security Authorization Facility (SAF) compliant and provides a user exit facility to give gateway installations a mechanism for passing control to installation-written code at user logon time. Security environments using the SAF interface include ACF2, RACF, and TOP SECRET.

In addition, Oracle Transparent Gateway for DB2 can be configured to pass DB2 statistics to the Oracle optimizer for improved query performance. Oracle Transparent Gateway for DB2 also provides customers with the ability to take advantage of DB2 parallel query operations.

Oracle Transparent Gateway for EDA/SQL

Oracle Transparent Gateway for EDA/SQL combines technology from Oracle Corporation and Information Builders. Oracle Transparent Gateway for EDA/SQL enables access for Oracle applications to the non-Oracle databases and file systems supported by the EDA/SQL Server. These include the following:

- VSAM
- IMS
- ADABAS
- TOTAL
- Teradata
- Infoman
- CA-IDMS
- SYSTEM 2000
- FOCUS
- DB2
- ISAM
- Datacom
- SUPRA
- Model 204
- QSAM

Support includes the EDA/SQL Server capability for SQL INSERT, UPDATE, and DELETE for IMS TM and IDMS data sources without the use of remote procedures.

2

Using the OS/390 Database Instance

The Oracle9*i* database server is implemented as a database service under OSDI. It can monitor events relevant to its operation, such as TSO logoff, abnormal terminations, and operator console commands.

This chapter describes basic information on installation, utilities, and utilities needed for using the Oracle9*i* for OS/390 server and contains the following sections:

Oracle Database Instance Overview

The Oracle database instance on OS/390 is implemented as a service under a given Oracle subsystem. This chapter discusses considerations for connecting local (TSO, batch, CICS, IMS) Oracle utilities or applications to database instances running on the OS/390 system. Connecting from UNIX System Services is discussed in Chapter 4, "Accessing Oracle9i Under USS".

Oracle9*i* database server users on OS/390 communicate with the local Oracle database instance using OS/390 cross memory services. This facility allows both data and program operation to cross address space boundaries in a secure and controlled manner.

Connecting to an Oracle Instance

Each TSO or batch user of an Oracle9*i* for OS/390 instance runs as a separate and autonomous address space. You can only access an Oracle instance after a valid connection is established between the user address space and the Oracle instance. A valid connection occurs when the Oracle instance accepts the logon user id and possibly a password provided by the user.

This section discusses considerations for connecting local TSO or batch Oracle utilities or applications to Oracle database instances running on the same OS/390 system. Such connections use OS/390 cross-memory facilities and do not involve the Oracle Net. However, since the Oracle database server logically views all client connections as network connections, Oracle network terminology and some of the mechanisms of Oracle Net play a role in local connections.

Enabling OS/390 Local Client Access

Oracle database server clients use a cross-memory protocol for connecting to database instances. The protocol is based on Oracle Net architecture.

Since Oracle's cross-memory protocol is based on Oracle Net, the formal mechanism for specifying a local target Oracle instance is to supply a name that is looked up in the client's tnsnames file (which is identified by the TNSNAMES DD statement). Applications supply the name by appending an "@" character followed by the name to the userid and password that are passed during an Oracle connect request. The entry in the tnsnames file contains an Oracle Net address string with PROTOCOL=XM and additional parameters identifying the target database service. The complete format of the PROTOCOL=XM Oracle Net address is described under "Cross-Memory Protocol Address" on page 2-2. Usually the target service is identified by its SID. Every service must have a SID that is unique throughout the OS/390 image. Even services that are defined in different Oracle subsystems cannot have the same SID if they are on the same OS/390 image.

It also is possible to append the complete Oracle Net address string directly to the userid and password to avoid using the tnsnames file. Oracle does not recommend this technique, however.

Cross-Memory Protocol Address

The formal Oracle Net address string for Oracle's cross-memory protocol can be specified using either of two methods:

- One method identifies the database instance by its SID:

`(ADDRESS=(PROTOCOL=XM)(SID=sid))`
where *sid* is the SID associated with the database instance.

- The other method uses the Oracle subystem and service names:

  `(ADDRESS=(PROTOCOL=XM)(SUBSYS=ssn)(SERVICE=srvn))`
  where *ssn* is the Oracle subsystem name, and *srvn* is the database service name.

  Oracle recommends using the SID form of address because it is simpler and because it avoids application dependence on the subsystem name.

Specifying Connections

There are four ways, including hardcoding an Oracle Net address on the connect string as described in the previous section, to specify a target instance. In descending order of precedence, the four ways are:

1. Specifying a tnsname entry which refers to the Oracle Net cross-memory protocol on the connect string.
2. Providing a DD statement (or equivalent TSO allocation) of the form:

   `//ORA@sid DD DUMMY`

   where *sid* matches the SID of the target service. Normally this is used only in batch or TSO applications.
3. A TWO_TASK environment variable that is set to an Oracle Net tnsnames-style name or to an explicit Oracle Net address string. If a name is specified, then the TNSNAMES DD in the client address space is opened and read to resolve the name to an Oracle Net address. If the Oracle Net address specifies PROTOCOL=XM, then the client is connected to the indicated OSDI service. (The Oracle Net address could also specify PROTOCOL=TCP, in which case the client would connect to a remote Oracle instance via Oracle Net as discussed in "Oracle Net Connect Descriptors for OS/390" in Chapter 10.) In terms of precedence, clients look for TWO_TASK after determining that no ORA@*sid* DD is allocated in the address space.
4. An ORACLE_SID environment variable is set to the SID of a database instance; the client is connected to that instance. Clients look for ORACLE_SID last, after determining that no TWO_TASK environment variable is set.

Installation Information

Installation-specific information is required to use the Oracle9*i* database server effectively. This information is available from your Oracle DBA.

Oracle9*i* Server and Library Names

The installation process creates several load and source module libraries you might need to access. For example, you might need information about the following:

- CMDLOAD library name

  All utilities, programs, and dynamically loaded modules for Oracle utilities are located in the CMDLOAD library. The person who installs the Oracle9*i* database server at your site determines this data set name.

  The examples presented in this guide assume the CMDLOAD library has the data set name `oran.orav`.CMDLOAD.

- MESG library name

  This library contains NLS data objects and all Oracle message modules.

- SQLLIB library name

  The SQLLIB library contains modules used in linking programs developed in high-level languages and IBM Assembler. It contains modules used by both Oracle Precompiler applications and OCI applications.

- SRCLIB library name

  The SRCLIB library contains sample JCL, program code, and scripts for Oracle tools, Programmatic Interfaces and Access Manager.

- OBJLIB library name

  The OBJLIB library contains the object module ()RASTBS) for liking Oracle call Interface programs that do not use the LONGNAME compiler option.

Refer to the *Oracle9i Enterprise Edition Installation Guide for OS/390* for more information on load and source module libraries.

Tool Syntax and Batch Procedures

Contact your Oracle DBA for the following information you need to use the Oracle utilities:

- Name and location of the Oracle cataloged procedures

The following factors influence the syntax used to run the Oracle utilities:

- Allocation of the Oracle CMDLOAD library to the STEPLIB DD statement in the TSO LOGON procedure
- Installation of TSO/E
- Allocation of the ORA$ENV DD statement
- Allocation of the ORA$LIB DD statement
- Allocation of the TNSNAMES DD statement
- Inclusion of Oracle CMDLOAD library in the system LINKLIST
- Installation of reentrant Oracle modules in the system link pack area (EPLPA and PLPA)

OS/390 Features

Use of Oracle utilities is subject to the normal conventions and features of the OS/390 operating system. The syntax of TSO commands, CLISTs, and batch JCL must be followed.

Using Oracle Utilities

You can invoke the Oracle utilities from TSO or you can use JCL procedures to submit a batch job.

Invoking Oracle Utilities from TSO

How you invoke Oracle utilities from TSO at your site might differ slightly from what is described in this guide because access standards and requirements differ among installations. Your DBA can provide you with the command syntax standards specific to your installation.

Call the Oracle utilities from TSO using one of the following:

- Running as a command processor
- Using the TSO CALL command

You control tool operation with CLISTs and the Interactive System Productivity Facility (ISPF) regardless of which technique you use. The technique used is largely determined by decisions made at your site during installation. The technique can also be affected by the number of active Oracle instances and the release levels of

the instances. Contact your DBA if you have any questions about which technique to use.

Note: You must allocate the MESG data set as ORA$LIB DD before invoking any utilities:

```
alloc file(ORA$LIB) da('orav.oran.MESG') shr reuse
```

Oracle Utilities as Command Processors

When the Oracle CMDLOAD library is concatenated to your STEPLIB or when its contents are placed in a linklist library, the Oracle utilities are accessible as command processors according to standard TSO conventions. The standard syntax is:

```
command [parm1][parm2]...[parmn][userid[/password[@tnsname]]] [options]
```

Oracle Utilities as CALL Programs

All the Oracle utilities are accessible through the TSO CALL command. Use the CALL command to specify operation of a program from a specific library. In the CALL command, specify the module to be run and any needed parameters. The CALL command syntax is:

```
CALL 'oran.orav.CMDLOAD(command)' '[parm1] [parm2]...[parmn] –
   [userid[/password[@tnsname]]] [options]'
```

Provide the CMDLOAD library name, the invocation command, one or more positional parameters followed by the user id and password, an optional connect string, and one or more options. The number of positional parameters and the options available are tool-specific. The single quotes are part of the command syntax and must be included as shown in the example.

The CALL command and other TSO commands can be continued from one line to the next using the normal TSO continuation syntax (a dash, –, to indicate continuation to the next line) and without regard for line formatting.

Oracle Utilities Accessed Through CLISTs

You may also access Oracle utilities using CLISTs. The CLIST can access the utilities as command processors or as called programs. The developer of a CLIST can perform preliminary allocations of files, check for error conditions, and generally ensure that access to the Oracle server proceeds according to installation standards.

If you have any questions about the approved method of calling any Oracle tool at your site, then contact your DBA.

Submitting a Batch Job

The Oracle installation process optionally creates batch JCL procedures you can use to call the Oracle utilities. The procedures at your installation might have different names than those in the following list if a procedure name suffix is selected during the installation. Check with your system administrator before attempting to use the following procedures.

Table 2–1 Batch JCL Procedures for OS/390

Procedure Name	Procedure Function
ORAC	calls the Pro*C Precompiler.
ORACB2	calls the Pro*COBOL Precompiler version 9.
ORACOB	calls the Pro*COBOL Precompiler version 1.
ORADBV	calls the DBVerify utility.
ORAEXP	calls the Export utility.
ORAFOR	calls the Pro*FORTRAN Precompiler.
ORAIMP	calls the Import utility.
ORALDR	calls SQL*Loader.
ORAOTT	calls the Object Type Translator.
ORAPLI	calls the Pro*PL1 precompiler.
ORARMN	calls the Oracle9*i* Recovery Manager utility.
ORASQL	calls SQL*Plus.

Sample Cataloged Procedure

The following example shows the ORASQL JCL procedure, which runs SQL*Plus:

```
//ORASQL   PROC  INDEX=oran,
//               LIBV=orav,
//               SYSOUT='SYSOUT=*'
//ORASQL   EXEC  PGM=SQLPLUS,REGION=4M
//STEPLIB  DD    DSN=&INDEX..&LIBV..CMDLOAD,DISP=SHR
//ORA$LIB  DD    DSN=&INDEX..&LIBV..MESG,DISP=SHR
//SYSOUT   DD    &SYSOUT,DCB=(LRECL=132,BLKSIZE=1320,RECFM=VB)
```

```
//SYSERR   DD    SYSOUT=*,DCB=(LRECL=132,BLKSIZE=1320,RECFM=VB)
//ORAPRINT DD SYSOUT=*
//* REQUIRES //SYSIN DD* STATEMENT
//* REQUIRES //ORA@sid DD DUMMY STATEMENT (ORACLE INSTANCE NAME)
```

The INDEX and LIBV symbolic parameters are used to form the STEPLIB and ORA$LIB data set names. The actual values for these parameters are established by the person who installed the Oracle9*i* database server. The cataloged procedures ensure the default values of these symbolic parameters conform to the actual names selected for the Oracle data sets at your installation.

In the example, the installed Oracle INDEX and LIBV parameters are `oran` and `orav`, respectively. These values result in the following data set name for the STEPLIB DD statement:

```
oran.orav.CMDLOAD
```

The SYSOUT DD statement contains a symbolic reference to &SYSOUT. This symbolic reference defaults to the string SYSOUT=*, which sets the SYSOUT class of this DD statement to the same class as MSGCLASS in the JCL JOB statement.

The procedure requires additional JCL to point to the correct Oracle instance. See earlier section 'Specifying Connections' for details. Some examples are:

Example 2–1 Example #1

```
//ORA@ORA1 DD DUMMY
```

Example 2–2 Example #2

```
//ORA$ENV     DD *
TWO_TASK=ORANAME
//TNSNAMES  DD *
ORANAME=(DESCRIPTION=
    (ADDRESS=(PROTOCOL=XM)(SID=ORA1))
)
```

Example 2–3 Example #3

```
 //ORA$ENV     DD *
 ORACLE_SID=ORA1
```

All three of the above examples illustrate accessing a local instance whose SID is ORA1.

Alternatively, the connection can also be specified using a tnsname alias on a userid and password supplied in the SYSIN file, or in the PARM field of the EXEC statement.

This procedure requires a SYSIN DD statement. The SYSIN DD statement can contain instream SQL and SQL*Plus statements, or it can point to a file containing such statements.

Each JCL procedure provided with the Oracle9*i* database server conforms to the basic structure illustrated by the ORASQL procedure.

Sample Batch Job

The following example JCL runs the ORASQL cataloged procedure. You must provide an appropriate JOB card at the beginning of the sample job. The example assumes the Oracle instance you are connecting to has ORA1 as its SID.

```
// ...     JOB   ...
//SQL EXEC ORASQL,
//    PARM='SCOTT/TIGER'
//ORA@ORA1 DD DUMMY
//SYSIN    DD DSN=SCOTT.ORACLE.SQL(SQLSTUFF),DISP=SHR
```

- User id SCOTT is used to connect to the Oracle instance.
- The ORA@ORA1 DD statement names the target instance, ORA1.
- The contents of member SQLSTUFF in data set SCOTT.ORACLE.SQL is run. This file must contain valid SQL and SQL*Plus statements.

Ask your DBA or systems programming staff for the location of these JCL procedures and the correct symbolic parameters that must be provided to the utilities. These procedures are installed in a library that is automatically searched by the JES2 or JES3 job entry subsystem. If you experience problems in running these procedures, then contact your Oracle DBA.

Using the Batch TMP

Run Oracle utilities in batch using TSO notation and conventions by running the TSO batch terminal monitor program (TMP) facility.

Exiting Utilities and Utilities in TSO

In response to an input prompt, a forward slash followed by an asterisk (`/*`) is used to exit any line mode Oracle9*i* for OS/390 tool or utility.

Attention Processing

For TSO applications, the Oracle subsystem establishes an attention processing subtask through the OS/390 STAX macro instruction. When the [Attn] or [PA1] key is pressed, the interrupted status is passed to the connected Oracle address space. The corresponding main task then performs the interrupt processing and returns to the application address space. The attention processing is removed when the disconnect function is called and the attention processing subtask is detached.

For both Oracle tools and customer-written applications running in TSO/E, the Oracle program interface code will attempt to establish an attention-processing subtask that uses the TSO/E STAX service.

This mechanism allows the user to interrupt an operation in the Oracle server by pressing the terminal [Attn] or [PA1] key. For this to work, the ORASTAX module must be loadable from JOBLIB/STEPLIB, a linklist library, or LPA. If ORASTAX cannot be loaded, attention processing is suppressed; no messages are issued to indicate this condition.

In addition to the above, certain Oracle tools or utilities recognize and process TSO/E attention if it occurs while the utility is active (that is; not during a server request). These are discussed in the relevant tool or utility sections.

Attention interrupt processing is specific to each tool. Chapter 9, "SQL*Plus" discusses the effect of attention processing.

OS/390 Environment Variables

Environment variables enable you to set certain options for each user. Specify environment variables in a sequential file or in a PDS member pointed to by the ORA$ENV DD statement.

Environment variables have the following syntax:

```
user_env_name = 'env_value'  *** comment ****
```

where:

`user_env_name`	is the environment variable name. This name can be expressed in uppercase, lowercase, or mixed case.
`env_value`	is the environment variable value you specify. If the value is enclosed by single quotes, then it is saved exactly as you specify it. If the value is not enclosed in single quotes, then the value is translated to uppercase and saved in uppercase.

`comments` is an optional comment. If comment is used, then '`env_value`' must be enclosed by quotes. If the first nonblank character is '*', '/', or '#', then the whole line is treated as a line comment.

The following environment variables are supported:

- Oracle NLS parameters as described in the *Oracle9i Database Globalization Support Guide*
- Oracle connection environment variables, ORACLE_SID and TWO_TASK
- OS/390-specific environment variables

ORACLE_SID, TWO_TASK, and the OS/390 environment variables are described in the following list.

CRTL_SPFENQ enables or disables the PDS enqueuing mechanism for Oracle utilities. For example:

```
CRTL_SPFENQ = 'yes|no'
```

The default is `yes`.

The enqueue mechanism used by Oracle utilities for performing simultaneous updates to a shared PDS is compatible with the method used by the ISPF editor. This enqueue mechanism affects PDS members opened with /DD/ or /DSN/.

CRTL_PDSWAIT alters the behavior of the PDS enqueue mechanism. When the enqueue is not immediately available, a `yes` places the user in a wait state. When the enqueue is not immediately available a `no` returns an error to the user. The default is `no`. Following is the correct syntax:

```
CRTL_PDSWAIT = 'yes|no'
```

NLS_LANG	specifies the national language in which Oracle utilities issue messages, format and display dates and numbers, sort text, and process character data at computers. The syntax is: NLS_LANG='language_territory.charset' The default setting is: NLS_LANG='AMERICAN_AMERICA.WE8EBCDIC1047' Customers who have built their databases with releases earlier than 7.3.2 and who have 'WE8EBCDIC37C' as their character set can specify the following to eliminate unnecessary conversion: NLS_LANG='AMERICAN_AMERICA.WE8EBCDIC37C' Consult with your Oracle DBA for information about which languages, character sets, and territories are supported at your site.
ORACLE_HOME	specifies the OS/390 USS directory path of the location where the USS components of Oracle9*i* for OS/390 are installed.
ORACLE_SID	specifies the SID of a local Oracle for OS/390 database instance that is used to connect a local Oracle for OS/390 client to a local Oracle database instance.
TWO_TASK	specifies the Oracle Net tnsnames-style name or the explicit Oracle Net address string that is used to connect a local Oracle for OS/390 client to a local or remote Oracle database instance.
TZ	specifies the time zone that is used within the Oracle JServer. If TZ is not defined, a default of GMT is used. The TZ specified should match one of the entries from the `$ORACLE_HOME/javavm/lib/tzmappings` file.

3

Oracle9*i* Utilities and OS/390 Files

Oracle utilities provide flexible file support within the OS/390 environment. Using file specification syntax and a special feature, Oracle utilities allow great flexibility in specifying and accessing OS/390 data sets and other types of files.

This discussion does not apply to Oracle utilities running in the UNIX System Services environment. For information on Oracle utilities running in the USS environment, see Chapter 4, "Accessing Oracle9i Under USS".

This chapter describes the interaction between OS/390 files and Oracle utilities. The following topics are discussed:

Using OS/390 Files

All Oracle utilities use OS/390 files. Every tool or utility accesses one or more input or output files. The term file refers to the logical unit that is the actual repository of the data the tool or utility reads from or writes to. A file can be an OS/390 data set, a TSO screen, the OS/390 operator console, an input or output spool file, or even a load module containing static input data.

Most OS/390 users are accustomed to utilities or applications that operate on files specified in DD statements or TSO ALLOC commands. Many TSO commands and some IBM utilities (such as IDCAMS) let you specify a file as a data set name, performing dynamic allocation automatically instead of requiring a pre-allocated

DD statement. With Oracle file support, you can also create new data sets without a DD statement and you can access devices (such as the operator console) OS/390 does not usually consider files.

The Oracle9*i* database server supports all non-VSAM file organizations except spanned records (RECFM=VS or VBS) and ISAM (DSORG=IS). For information about the VSAM files that comprise the Oracle control, database server, and redo log files, refer to the *Oracle9i Enterprise Edition System Administration Guide for OS/390*.

General Notation for Specifying Files

All Oracle9*i* for OS/390 utilities recognize a uniform notation for specifying files. The general form of this notation (called a file specification or filespec) is shown in the following example:

```
[/pathname/]data
```

where:

`pathname`	is one of several reserved names specifying the type of filespec.
`data`	is additional information or specific file related parameters.

The following string indicates a file identified by DDname INFILE in the present job step or TSO session:

```
/DD/INFILE
```

File Specification Types

You can specify files in the following ways:

- By DDname
- By full data set name
- By unqualified data set name
- By PDS member (as a control or parameter file)

Summary of Filespecs

The following table summarizes the types of file specifications you can use with Oracle utilities.

Table 3–1 Summary of Filespecs

Syntax	What is Accessed
`dsname`	The fully-qualified sequential data set that results after TSO prefix and filetype suffix extensions are performed.
`dsname(mem)`	Member `mem` of the fully-qualified PDS that results after TSO prefix and filetype suffix extensions are performed.

`/DD/ddname`
or
`DD:ddname`
	data set or spool file designated by the corresponding DD statement or ALLOC command.
	or
	The Language Environment equivalent

`/DD/ddname(mem)`
or
`DD:ddname(mem)`
	Member `mem` of the PDS designated by the corresponding DD statement or ALLOC command.
	or
	The Language Environment equivalent

`/DSN/dsname`
or
`//dsname(mem)`
	The fully-qualified sequential data set `dsname`.
	or
	The Language Environment equivalent

`/DSN/dsname(mem)`
or
`//dsname(mem)`
	Member `mem` of the fully-qualified PDS `dsname`.
	or
	The Language Environment equivalent

The file specifications and the notation for using them are described in the following sections:

- Specifying Files by DDname (/DD/)
- Specifying Files by Full Data Set Name (/DSN/)
- Specifying Files by Unqualified Data Set Name

Specifying Files by DDname (/DD/)

To specify a file by DDname to an Oracle tool use the following string, where `ddname` is the one-character to eight-character DDname:

```
/DD/ddname
```

The DDname must be allocated to the job step or TSO session before the file is opened. In a TSO session, an ALLOC command must be issued, usually before the

tool is called. (During a SQL*Plus session, you can use the HOST command to issue a TSO ALLOC command for a file used by a subsequent SQL*Plus command such as START or SAVE.) In a batch job, a matching DD statement must be included in the JCL.

You can specify a member of an OS/390 PDS as part of the /DD/ notation. The member name must be provided in parentheses immediately after the DDname, as shown in the following example:

```
/DD/MYPDS(SQL2)
```

The DD statement for batch (or TSO ALLOC) specifies the PDS name but no member name, as shown in the following example:

```
//MYPDS DD DISP=SHR,DSN=PAYROLL.SQLLIB.CNTL
```

Usage Notes The following usage notes are for specifying files by DDname:

- When an existing PDS member is specified for output, the current contents of the member are completely replaced by what is written. If a nonexistent member name is specified for output, then a new member with that name is created.
- When a JES spool file is used by an Oracle tool, the file must be specified by DDname. The corresponding DD statement or ALLOC command indicates a SYSOUT class (output spool files) or DD * (input spool files, only used in batch jobs).

See the SPOOL OUT command in "SPOOL OUT" on page 9-10 for an example specifying a filename with /DD/.

Specifying Files by Full Data Set Name (/DSN/)

You can specify a file by OS/390 data set name to an Oracle tool using the following syntax, where `hlq.dsname` is the fully-qualified data set name:

```
/DSN/hlq.dsname
```

This notation gives access to a sequential OS/390 data set with the fully-qualified data set name. The Oracle9*i* database server dynamically allocates the data set. You do not need to provide a DD statement or a TSO ALLOC command.

Access a member of a PDS by including a member name in parentheses immediately after the data set name, as in the following example:

```
/DSN/ORACLE.OSDI.CNTL(ORASQL1)
```

In this case, the member ORASQL1 of PDS ORACLE.OSDI.CNTL is read or written.

Usage Notes The following usage notes are for specifying files by full data set name:

- If the file is being used for input, then the data set must exist and be cataloged; otherwise, an error is reported.
- If the file is being used for output, then the existing data set of the specified name is overwritten. If the data set does not exist, then it is created through dynamic allocation with your system's default space allocation.
- When an existing PDS member is specified for output, the current contents of the member are completely replaced by what is written. If a nonexistent member name is specified for output, then a new member with that name is created.
- If an output file using PDS member notation specifies a data set name that does not exist, then the data set is created dynamically in the same manner as a sequential data set. However, because OS/390 has no default space allocation for PDSs, the Oracle9*i* server inserts its own default of one track primary, one track secondary, and one PDS directory block. This default is not appropriate for most PDS applications.

See the start file parameter in "Running Under TSO" on page 9-2 for an example of the /DSN/ notation.

Specifying Files by Unqualified Data Set Name

You can specify a file without a pathname to an Oracle tool. If a file specification does not have a pathname (it does not begin with a slash), then it is treated as a partial data set name that is subject to extension.

Most Oracle utilities also extend unqualified data set names by appending a suffix to the right of the name. The suffix, often called a filetype extension, is generally an acronym or mnemonic describing the type of data or records in the file. For example, files read by the SQL*Plus START command, which normally contain SQL*Plus commands or SQL statements, have a filetype extension of SQL. "Filetype Suffixes" on page 3-8 provides a list of the filetype extensions used by Oracle utilities.

Unqualified data set file names, like fully-qualified names, can refer to PDS members by including the member name in parentheses immediately after the name. The addition of the filetype extension does not affect the member name because the suffix is added before the member name.

Usage Notes The following usage notes are for specifying files by unqualified data set name:

- Only file names specified without a pathname are subject to extension processing.
- Extension processing is further controlled by the File Name/Attribute Augmentation Facility (FNA).

Examples To illustrate prefix and suffix extension of unqualified data set names, assume a TSO user, whose PROFILE PREFIX is GJONES, calls SQL*Plus and then enters the following command:

```
START MYSQL2
```

Because there is no pathname prefix, MYSQL2 is considered an unqualified data set name and is extended (at both ends) to become GJONES.MYSQL2.SQL. The following command produces the same result:

```
START /DSN/GJONES.MYSQL2.SQL
```

The next example illustrates unqualified file specification with a member name:

```
START MYSQLLIB(TEST2)
```

This name is extended to GJONES.MYSQLLIB.SQL (TEST2).

Redirecting Standard Files and Parameters

You can redirect standard files and parameters to specify a nondefault file for the duration of the associated TSO command or job step. To specify redirection, use one of the following symbols followed by a file specification from the command parameter line in TSO or the PARM field of the EXEC statement in batch:

<	for input
>	for output
>>	to append output
?	for error output
??	to append error output
++	for parameters

Redirecting Files

Any of the standard input, output, and error files can be redirected with these symbols. While the redirection parameters can be mixed with other command or utility parameters, Oracle Corporation recommends grouping these parameters at the beginning or end of the command parameters or PARM field for clarity. The following example redirects the input file to the INFIL DD statement.

```
//STEP1 EXEC PGM=PROC,PARM='</DD/INFIL'
//INFIL DD *
```

If an output file specified by a data set name does not exist, then the Oracle application dynamically allocates it using the installation or IBM default space and unit specifications. If the output file does exist, then the Oracle application normally overwrites any existing contents.

You can append output to a data set by doubling the redirection prefix symbol. If the data set does not already exist, then it is created. For example, the following command adds error messages to the end of the data set named `tsoprefix`.MY.ERRORS:

```
??MY.ERRORS
```

Redirecting Parameters

To use parameter redirection, imbed the following string in the OS/390 parameter field, where `filespec` is a valid Oracle for OS/390 file specification:

```
++filespec
```

When the Oracle tool or utility encounters this in the parameter field, it scans the data in the file specified by `filespec` and processes it as if it is included in the OS/390 parameter field. At end-of-file (EOF), the Oracle tool or utility continues scanning the OS/390 PARM field at the end of `filespec`.

The following example uses a parameter redirection operator:

```
//STEP1 EXEC PGM=PROCOB18,PARM='++/DD/SYSPARM HOST=COBOL'
//SYSPARM  DD  *
  INAME=/DD/SYSIN
  ONAME=/DD/SYSPUNCH
  LNAME=/DD/SYSPRINT
  USERID=SCOTT/TIGER
/*
```

In this example, the Oracle Pro*COBOL Precompiler processes all the data (including INAME=. . . . . USERID=SCOTT/TIGER) as if it is passed in the OS/390 parameter field. You can specify parameter redirection anywhere in the OS/390 parameter field and it is implemented at that point.

Note: The ++ filespec does not support the use of redirection parameters.

Filetype Suffixes

The contents of Oracle files can be identified by the filetype suffix. A filetype suffix is supplied by each Oracle tool for each file used. The suffix is a short mnemonic or acronym identifying the file contents. The suffixes in Table 3–2 identify the different types of Oracle files:

Table 3–2 Filetype Suffixes for Oracle Utilities

Suffix	Filetype
BAD	Reject record output from SQL*Loader.
BUF	An input and output save file in SQL*Plus.
CTL	A control file input to SQL*Loader.
DAT	An input file to SQL*Loader.
DMP	The output from the Export utility.
DSC	Discarded record output from SQL*Loader.
LOG	A log output file from SQL*Loader.
LST	A print or listing file from SQL*Plus.
PKH	Oracle9*i* Text PL/SQL header files used during installation.
PLB	A PL/SQL binary file.
SQL	An input file to SQL*Plus.
XLT	Oracle9*i* Text file.

File Name/Attribute Augmentation (FNA) Facility

Many utilities support the full range of naming syntax described in "Using OS/390 Files" on page 3-1. Some utilities require file names in a simpler format or shorter

length than is preferable in OS/390. If the full range of naming syntax is not available, then you can take advantage of FNA name manipulation to convert a simple name string to a larger, more complex file name string.

The FNA facility provides automated user-defined manipulation of file names beyond the scope of normal OS/390 file name and extension processing. Before a tool opens a file, it calls FNA processing. The tool supplies FNA with the base file name and filetype suffix. FNA builds the final file name from this input, using the file syntax array (FSA) data and a set of rules. The final file name string, which is essentially unrestricted in size and syntax, is returned to the tool. The tool then proceeds with opening and reading or writing the file.

Only file names specified without a pathname (without a `/.../` prefix) are subject to extension processing.

With FNA, you can do the following:

- Access OS/390 data set names that have relatively complex syntax (such as PDS).
- Overcome logical ambiguities imposed by nonextendable file names (those prefixed with a pathname).
- Select JCL-like parameters (such as DCB attributes and DASD space allocations) for specific types of files written by Oracle utilities.

A small set of FNA controls is distributed with Oracle9*i* for OS/390. These controls ensure the use of specific DCB attributes with several types of output files. You can modify, eliminate, or expand these controls.

FNA Controls and Operations

FNA is controlled by the file syntax array (FSA), a logical table containing specifications for modifying file names and adding JCL-like keywords before a file is actually opened. The FSA contents are determined by the following:

- A set of default FSA entries distributed with the product.
- An optional user-defined control file that adds more FSA entries or alters the distributed default entries.

FNA file name manipulation is controlled by the FSA contents and the following specifications:

base file name	A character string, usually (though not always) supplied by the user.

filetype suffix	A character string supplied by the Oracle tool, which identifies the type of file being used.
TSO PROFILE PREFIX	A character string that might have been set by the PROFILE PREFIX command (this command must be issued before calling the Oracle tool).

You can supply the base file name in the following ways:

- In the parameters specified when a tool is called
- In response to a prompt from the tool
- In a subcommand of the tool (like the SQL*Plus SPOOL command)

FSA Table

The logical table of FSA data used by FNA is comprised of three columns. The first column contains filetype suffixes. Each filetype for which FSA data is present appears in this column once. The second column contains character strings that designate construction of the final file name for the corresponding filetype. The third column contains character strings of JCL-like keywords. These are added to final file names for the corresponding filetype and are only effective when the file is opened for output.

FNA File Name Construction

To construct a file name, FNA substitutes characters as appropriate, then extends the name accordingly.

Substitution

The FNA file name construction process copies all characters from the modification string except plus signs exactly as given. FNA uses the following substitution for these special characters:

+	is replaced with the user-supplied base file name.
*	when used as the high level qualifier, is replaced with the current TSO PROFILE PREFIX value when running in the foreground, and the TSO login ID must be used when running in batch mode.

Construction Procedure

FNA takes the base file name and filetype suffix as input and uses the following procedure to construct the final file name:

1. The FSA is searched for an entry with a filetype matching the filetype suffix provided. If a matching entry is found, then FNA proceeds with Step 2 through Step 4. If no matching entry is found, then FNA uses the unmodified base file name and proceeds with Step 4.
2. If the FSA contains a file name string, then that name is modified. If the FSA does not contain a name string, then the base (user-supplied) file name is modified.

   To modify the name, first the base file name is substituted wherever the plus sign appears in the FSA file name string, and then the TSO PROFILE PREFIX is substituted wherever an asterisk appears. If no PREFIX is defined, then each asterisk is replaced by a null string.
3. If the FSA entry for the filetype contains a non-null attribute string, then a colon (`:`) and the attribute string are appended to the file name (which might have been modified by Step 2).

   The colon is a delimiter to separate the attribute keywords from the file name. The attribute string is the third column of the FSA entry, where JCL-like attributes are specified. The addition of attributes is only meaningful for files that undergo dynamic allocation. FNA attributes for /DD/ files have no effect.
4. If the file name does not begin with a path prefix (`/`) or apostrophe (`'`), then the following steps are taken:

   - A period and the filetype suffix are added on the right. The filetype suffix is added so any PDS member name or attributes in parentheses are kept on the right end of the file name.
   - If attributes are specified by the user in addition to those specified in the FSA, then the user's attributes are attached to the name following those from the FSA. If the user specifies attribute keywords that duplicate those in the FSA, then the user's attributes are used.
   - The following string is attached to the beginning of the name, where `prefix` is your TSO PROFILE PREFIX:

     `/DSN/prefix`

     If the PROFILE PREFIX is not set or you are not running under TSO, then only /DSN/ is attached to the name.

FNA processing is complete and the file name is returned to the tool, which opens and then reads or writes the file.

FNA Example

Suppose TSO user GJONES (with PROFILE PREFIX set as GJONES) has issued the following SQL*Plus command, which specifies a base file name of MYSQL2:

```
START MYSQL2
```

SQL*Plus associates a filetype of SQL with the file used in the START command. Assume also that the following entry is in the FSA:

```
FSA (
  FTYPE(SQL)
  FNAME('SQLLIB(+)')
  FATTR('VB,LRECL=240')
  )
```

The following sample shows an FNA process:

1. Locate the FSA entry

   This locates the FSA entry for the filetype SQL (case is ignored).

2. Perform the substitution

   This processes the file name string SQLLIB(+), substituting the base file name MYSQL2 for the +, resulting in the following:

   ```
   SQLLIB(MYSQL2)
   ```

3. Add the attribute string

   This step appends a colon (`:`) and the attribute string. The section "Attribute Keywords" on page 3-16 discusses the attribute string. The result is the following name:

   ```
   SQLLIB(MYSQL2):VB,LRECL=240
   ```

Since the name does not begin with a path prefix or apostrophe, the following steps are completed:

1. Add the suffix

   Attaches a period and a filetype suffix to the end of the name, before the member name in parentheses. The result is the following name:

   ```
   SQLLIB.SQL(MYSQL2):VB,LRECL=240
   ```

2. Add the user's attributes

No additional attributes are specified with the SQL*Plus START command, so the name remains unchanged.

3. Add /DSN/ and the TSO PREFIX

   This step attaches the /DSN/ pathname and the TSO PROFILE PREFIX to the beginning of the name, resulting in:

   ```
   /DSN/GJONES.SQLLIB.SQL(MYSQL2):VB,LRECL=240
   ```

This file name string is passed back to SQL*Plus and opened. Because the START command opens the subject file for input, the attribute data is ignored and the data set's existing DCB attributes are observed. A member called MYSQL2 in the PDS named GJONES.SQLLIB.SQL is opened for the user's MYSQL2 specification.

You can use FNA to provide additional flexibility in managing the interaction between Oracle utilities and OS/390 files by controlling the FSA entries. The section "User-Defined FNA Control Files" describes additional examples of FSA entries and FNA processing.

Default FSA Entry

The distributed Oracle product contains the default FSA entry listed in Table 3–3. For all other filetypes, the defaults are null strings, which means FNA supplies no special name or attributes.

Table 3–3 Oracle Product Default FSA Entry

Filetype	FNAME	FATTR
DMP	(null)	FB,LR=4096

The DMP entry sets DCB attributes required for Export data.

The FSA default entry can be overridden, unless otherwise noted in Table 3–3, by supplying an FNA control file with an entry for the corresponding filetype with a string specified for the item (FNAME or FATTR) to be overridden.

User-Defined FNA Control Files

All Oracle utilities use the default FSA entries every time the Oracle9*i* database server is called. You cannot permanently modify the default entries. However, you can define a control file that includes FSA entries for a specific invocation of an Oracle tool.

An FNA control file has the following uses:

- Add FSA entries for suffixes not included in the default FSA, including specifications for modifying file names and appending output attributes.
- Change the default FSA specifications for modifying file names and appending attribute data. These changes can include eliminating the default specifications entirely.
- Add specifications for modifying file names to a default entry that only appends attribute data, or add specifications for appending attribute data to an entry that only modifies file names.

Specifying an FNA Control File

A user-defined FNA control file always has the following name:

```
/DD/ORA$FNA
```

You must allocate this file to an ORA$FNA DD statement in your JCL or TSO session. The ORA$FNA DD statement must refer to one of the following types of data sets:

- A sequential data set with fixed or variable length records
- An instream (DD *) data set
- A PDS member with the member name specified in the DD statement

If an ORA$FNA DD statement is present when a utility is called, then the utility refers to the FNA control file specified in the DD statement. If an ORA$FNA DD statement is not present, then FNA uses the default FSA entries.

Creating an FNA Control File

The FNA control file contains FSA entries specified with EBCDIC characters. Any number of FSA entries can be included. An FSA entry can be continued over multiple lines; no special continuation indicator is required. Comments, surrounded by `/*` and `*/`, can be included at any point that permits a blank or end of record. The control file records must not contain sequence numbers.

FSA Entries

FSA entries have the following format:

```
FSA (FTYPE(suffix-type)
     FNAME('name-string')      /*<optional>*/
```

```
FATTR('attribute-string')  /*<optional>*/
)                         /*<required>*/
```

where:

suffix-type
is a one-character to eight-character string indicating a file name suffix. Any file name with this suffix is substituted by the value provided by the FNAME keyword that follows.

Do not enclose this string in single or double quotes. The string is converted to uppercase characters. The first blank space or right parenthesis, , is considered the end of the string. For example:

FTYPE(SQL)

name-string
is an alphanumeric string indicating the file name to substitute for files that have the suffix specified in FTYPE. The FNAME keyword is optional.

Because FNAME is optional, the value for the omitted keyword is the value in the default FSA distributed with the Oracle9*i* database server, if any. If there is no default entry in the default FSA, then an omitted FNAME is assumed to be a null (empty) string.

Enclose the substituted file name in single or double quotes; blanks within the quotes are recognized. For example:

FNAME('/DSN/*.ORACLE.+.SQL')

If the substituted file name needs more than one line in the control file, then extend the value to the end of the line and continue in the first position of the next line.

If the TSO PREFIX is GJONES and the user supplies a base file named MYSQL, then FNA processing resolves this name string to:

/DSN/GJONES.ORACLE.MYSQL.SQL

If there is no TSO PREFIX, then FNA processing resolves this string to the invalid file name string:

/DSN/.ORACLE.MYSQL.SQL

This particular FNAME is only usable with a TSO PROFILE PREFIX.

attribute-string
is the attribute data added to the substituted file name. The FATTR keyword is optional.

The attribute data can be in the form of a keyword list separated by commas and enclosed in single or double quotes. Do not include blanks. For example:

FATTR('VB,LRECL=255')

Because this keyword is optional, the value for the omitted keyword is the value in the default FSA distributed with the Oracle9*i* database server, if any. If there is no default entry in the default FSA, then an omitted FATTR is assumed to be a null (empty) string.

Refer to "Attribute Keywords" for information about the correct syntax for this keyword value.

Usage Notes The following usage notes are for FSA control file entries:

- Include any character you want added to the base file. These characters are added exactly as specified in the string.
- Use a plus sign, +, to indicate where to substitute the user-supplied base file name.
- When the utility is run with PROFILE PREFIX, use an asterisk (*) to indicate where to substitute the TSO PROFILE PREFIX in the file name. If the utility is run with PROFILE NOPREFIX, then the asterisk is ignored.

  To use the TSO prefix substitution function in a batch job, run the job under the control of the batch monitor program IKJEFT01. Use the TSO PROFILE PREFIX command in your SYSIN data stream to set the appropriate TSO prefix before calling the utility.
- To eliminate a default FSA entry, specify an empty string with a pair of adjacent single or double quotes.

Attribute Keywords

Attribute data that can be added to a file name includes the following examples with the equivalent JCL parameter. Keyword characters in lowercase can be omitted.

Table 3–4 Attribute Keywords and JCL Parameter Equivalents

Keyword	JCL Parameter Equivalent
BLKsize	DCB=BLKSIZE=n.
BLOck	indicates the space is allocated in blocks; it cannot be used with the CYL or TRK keywords.
CYL	indicates space is allocated in cylinders; it cannot be used with the BLO or TRK keywords.
DIrectory	indicates the number of directory blocks for a PDS.
LRecl	DCB=LRECL=n.
OLD	DISP=OLD (exclusive DSN enqueue).
PRIMary	indicates primary space allocation.
SECondary	indicates secondary space allocation.
SHR	DISP=SHR (shared DSN enqueue).
TRK	indicates space allocation is in tracks; it cannot be used with the BLO or CYL keywords.
UNIT	indicates the name of the allocation unit; it can be up to eight characters.
VOLume	indicates the specific volume of allocation; it can be up to six characters.

The following keywords indicate DCB record format:

Table 3–5 DCB Record Format Keywords and JCL Parameter Equivalents

Keyword	JCL Parameter Equivalents
F	RECFM=F
FA	RECFM=FA
FB	RECFM=FB
FBA	RECFM=FBA
FBS	RECFM=FBS
FBAS	RECFM=FBAS
V	RECFM=V
VA	RECFM=VA

Table 3–5 (Cont.) DCB Record Format Keywords and JCL Parameter Equivalents

Keyword	JCL Parameter Equivalents
VB	RECFM=VB
VBA	RECFM=VBA

FNA Control File Error Handling

Most logical or syntax errors in the FNA control file are detected when the Oracle utility initializes. Error messages are displayed in the standard error file, which is normally the screen in TSO (if PROFILE WTPMSG is set) or the OS/390 console (if PROFILE NOWTPMSG is set). In batch, the SYSERR DD statement controls the output destination. In general, an erroneous entry is ignored and utility operation proceeds as though the entry is not specified.

Errors within the quoted FNAME or FATTR strings, such as invalid name syntax or improper attribute keywords or values, are not detected until the utility opens the associated file. These errors normally result in descriptive messages written to the standard error file. Because such errors usually make it impossible to open the associated file, utility operation typically halts in this situation.

FSA Keyword Usage Notes

When you use an FNA control file to override a default FSA entry, the string you specify for FNAME or FATTR completely replaces the string in the default entry. Be careful to add attributes using FATTR; a default attribute not replicated in the control file FATTR string is lost. For example, the DMP filetype has default FATTR of FB and LR of 4096 to meet the requirements of Export. To force the DMP file to a specific allocation unit, you can use an FNA control file containing:

```
FSA(FTYPE(DMP) FATTR('UNIT=SYSTSO'))
```

Because the override FATTR string replaces the default, the RECFM with FB and LRECL with 4096 attributes are lost in this sample specification, resulting in an unusable Export file. To prevent this, you must respecify the needed defaults along with the attribute being added, as in the following example:

```
FSA(FTYPE(DMP) FATTR('UNIT=SYSTSO,FB,LR=4096'))
```

DCB attributes specified through FATTR, whether defaults or from an FNA control file, replace corresponding attributes that might already be associated with an OS/390 data set.

When writing to pre-allocated data sets with DCB attributes you want to retain, do not use an FATTR specification, as shown in the following example:

```
FSA ( FTYPE(SQL) FNAME('/DD/SQLS') )
```

Attributes supplied for a file specified with a /DD/ pathname generally have no effect. In this case, attributes must be specified on the DD statement or in the TSO ALLOC command, or must already exist (for an existing data set).

Files opened for output default to DISP=OLD. If you want to allow for concurrent use of PDSs, then you must override this default by including an FATTR string of SHR in the FNA control file.

Examples Using FNA

The examples in the following sections illustrate sample FNA control file specifications for:

- Partitioned data sets
- Shared libraries
- Group libraries

Example 1 Partitioned Data Sets

In the following example, assume the installation wants each user to have a PDS that stores the SQL files created and run by each user. The installation sets up an FNA control file containing the following entry:

```
FSA( FTYPE(SQL)/* SQL*PLUS */
FNAME('SQLS(+)'))
```

Assume a user with a TSO PREFIX of GJONES has an ORA$FNA DD statement pointing to the FSA in the previous example. When the user issues the SQL*Plus START command with PAYROLL as the argument, GJONES.SQLS.SQL(PAYROLL) becomes the file name and is used for the SQL file.

Without this FSA entry, the system defaults to file name GJONES.PAYROLL.SQL for the SQL file.

This example emphasizes two points:

- The general defaults for naming files are implemented in the default FSA, which has the following format:

  ```
  FSA   (
  ```

```
        FTYPE(suffix)
        FNAME('+')
        )
```

- You can use the FNA facility to specify simple names that are treated as members of PDSs.

Example 2 Shared Libraries

In this example, assume the installation wants shared libraries for SQL files. The installation sets up the following FSA control file:

```
FSA (
     FTYPE(SQL)
     FNAME('/DSN/SYS5.SQL(+)')
     )
```

Assuming the same user described in "Partitioned Data Sets" on page 3-19, the SYS5.SQL(PAYROLL) becomes the shared library name and is used for the SQL file.

The pathname /DSN/ preceding the data set name suppresses the extension processing normally performed under OS/390. Both /DSN/ and /DD/ can be used in the FSA.

Example 3 Group Libraries

In the following example, assume an installation groups users, and each group is identified by a TSO PREFIX. This FSA entry provides the necessary names for a library shared within a group:

```
FSA( FTYPE(SQL)
     FNAME('/DSN/*.SQL(+)')
```

Assume a user with a TSO PREFIX of GROUP2 uses the SQL*Plus command START PAYROLL and has an ORA$FNA DD statement pointing to the previous FSA. Group2.SQL(PAYROLL) becomes the shared group library name and is used for the SQL file.

Although the Oracle9*i* server dynamically creates these data sets, Oracle Corporation recommends the installation pre-allocate all data sets referenced by FNA. Pre-allocating the data sets that are referenced by FNA ensures your requirements are met. The default values selected by FNA might not be appropriate for your requirements.

4

Accessing Oracle9*i* Under USS

This chapter explains how to run selected Oracle utilities under OS/390 UNIX System Services (USS). Information about implementing your own customer-written Oracle applications so they run under OS/390 UNIX System Services in included at the end of this chapter. The following topics are discussed in this chapter:

OS/390 UNIX System Services Overview

OS/390 UNIX System Services is similar to other UNIX environments, and the Oracle9*i* utilities built for OS/390 UNIX System Services behave like their UNIX counterparts with very few, if any, differences.

When the guide describes Oracle utilities running under OS/390 UNIX System services, it assumes that you have some familiarity with UNIX (whether it is OS/390 UNIX System Services or some other version). This guide describes what you have to do to be able to run any of the Oracle9*i* utilities designed for OS/390 UNIX System Services and it highlights some differences between using Oracle utilities in the traditional OS/390 environment and in the OS/390 UNIX System Services environment.

Oracle Database Instance Overview

The Oracle database instance on OS/390 is implemented as a service under a given Oracle subsystem. This chapter discusses considerations for connecting local USS Oracle utilities or applications to Oracle database instances running on the OS/390 system.

Oracle9*i* database server users on OS/390 communicate with the local Oracle database instance using OS/390 cross memory services. This facility allows both data and program operation to cross address space boundaries in a secure and controlled manner.

Connecting to an Oracle Instance

Each USS user of an Oracle9*i* for OS/390 instance runs as a separate and autonomous address space. You can only access an Oracle instance after a valid connection is established between the user address space and the Oracle instance. A valid connection occurs when the Oracle instance accepts the logon user id and possibly a password provided by the user.

This section discusses considerations for connecting local USS Oracle utilities or applications to Oracle database instances running on the same OS/390 system. Such connections use OS/390 cross-memory facilities and do not involve Oracle Net. However, since the Oracle database server logically views all client connections as network connections, Oracle network terminology and some of the mechanisms of Oracle Net play a role in local connections.

Enabling OS/390 Local Client Access

Oracle database clients use a cross-memory protocol for connecting to database instances. The protocol is based on Oracle Net architecture.

Since Oracle's cross-memory protocol is based on Oracle Net, the formal mechanism for specifying a local target Oracle instance is to supply a name that is looked up in the client's tnsnames file (`tnsnames.ora`). Applications supply the name by appending an "@" character followed by the name to the userid and password that are passed during an Oracle connect request. The entry in the tnsnames file contains an Oracle Net address string with PROTOCOL=XM and additional parameters identifying the target database service. The complete format of the PROTOCOL=XM Oracle Net address is described under "Cross-Memory Protocol Address" on page 4-3. Usually the target service is identified by its SID. Every service must have a SID that is unique throughout the OS/390 image. Even services that are defined in different Oracle subsystems cannot have the same SID if they are on the same OS/390 image.

It also is possible to append the complete Oracle Net address string directly to the userid and password to avoid using the tnsnames file. Oracle does not recommend this technique, however.

Cross-Memory Protocol Address

The formal Oracle Net address string for Oracle's cross-memory protocol can be specified using either of two methods:

- One method identifies the database instance by its SID:

  ```
  (ADDRESS=(PROTOCOL=XM)(SID=sid) [(STAX=yes)])
  ```

  where *sid* is the SID associated with the database instance.

- The other method uses the Oracle subsystem and service names:

  ```
  (ADDRESS=(PROTOCOL=XM)(SUBSYS=ssn)(SERVICE=srvn))
  ```

  where *ssn* is the Oracle subsystem name, and *srvn* is the Oracle database service name.

Oracle recommends using the SID form of address because it is simpler and because it avoids application dependence on the subsystem name.

Specifying Connections

There are three ways, including hardcoding a tnsname entry on the connect string as described in the previous section, to specify a target instance. In descending order of precedence, the three ways are:

1. Specifying a tnsname entry which refers to the Oracle Net cross-memory protocol on the connect string.
2. A TWO_TASK environment variable that is set to an Oracle Net tnsnames-style name or to an explicit Oracle Net address string.
3. An ORACLE_SID environment variable is set to the SID of a database instance; The client is connected to that instance. Clients look for ORACLE_SID last, after determining that no TWO_TASK environment variable is set.

Storing Connection Information

You can use three general methods to store connection information:

1. In a `tnsnames.ora`
2. In an LDAP server pointed to by `sqlnet.ora` and `ldap.ora`

3. In an ONAMES server pointed to by `sqlnet.ora`

All files are located in $TNS_ADMIN if specified or in `$ORACLE_HOME/network/admin`. For a complete discussion on connect strings to remote servers, see the *Oracle9i Net Services Administrator's Guide* for a discussion of specifying a local connection, see Chapter 2, "Using the OS/390 Database Instance".

Break Processing

Use of the cross-memory (XM) protocol from a tool or application in USS causes a thread to be created for break signal (Control-C) processing. This allows the user to interrupt an in-progress server request similar to the attention subtask processing provided in TSO.

Due to a limitation of USS, the presence of the break-handling thread precludes using the fork() system call. The spawn() system call can be used instead. If your application specifically requires fork(), you must use TCP/IP protocol rather than XM.

Running an Oracle Utility on OS/390 Under USS

The following sections describe the environment variables needed to run Oracle utilities under UNIX System Services.

Environment Variables

OS/390 UNIX System Services, like other UNIX systems, supports the use of environment variables. These are character values that are maintained by the UNIX shell and made available to any program that asks for them. The various values are given a name, to make it easier to reference them. UNIX programs typically require that one or more environment variables be properly set before the program is run. Oracle9*i* programs are no different. This section describes the key environment variables that can be used with Oracle9*i* utilities that run under OS/390 UNIX System Services.

ORACLE_HOME

When an Oracle product is installed on OS/390 UNIX System Services, it is placed into a home directory. The directory name must be defined by the ORACLE_HOME environment variable before any Oracle product is run. For example, assuming the Oracle home directory is `/oracle/v920`, the following statement would set the variable ORACLE_HOME:

```
export ORACLE_HOME=/oracle/v920
```

Shell scripts are often used to set the value of ORACLE_HOME and other environment variables that need to be specified. The actual value of ORACLE_HOME must be provided by your Oracle system administrator.

TNS_ADMIN

The file `tnsnames.ora` contains tnsnames entries which could be used by Oracle utilities or user-written programs as connect string specifications. Its default location is `$ORACLE_HOME/network/admin` and can be overridden by using the TNS_ADMIN environment variable. Refer to the *Oracle9i Net Services Administrator's Guide* for further details.

TWO_TASK

A TWO_TASK environment variable is set to an Oracle Net tnsnames-style name or to an explicit Oracle Net address string. If a name is specified, then the `tnsnames.ora` file is opened and read to resolve the name to an Oracle Net address. The `tnsnames.ora` file location may be specified in $TNS_ADMIN, or is in `$ORACLE_HOME/network/admin` by default. If the Oracle Net address specifies PROTOCOL=XM, then the client is connected to the indicated database service. (The Oracle Net address could also specify PROTOCOL=TCP, in which case the client would connect to a remote Oracle instance via OSDI Network service as discussed in Chapter 10, "Oracle Net".) In terms of precedence, Oracle clients look for TWO_TASK after determining that no ORA@*sid* DD is allocated in the address space.

ORACLE_SID

An ORACLE_SID environment variable is set to the SID of an OSDI-managed database instance: OSDI connects the client to that instance. Oracle clients look for ORACLE_SID last, after determining that no TWO_TASK environment variable is set.

NLS_LANG

Oracle9*i* includes comprehensive National Language Support (NLS) that transparently handles all necessary conversions between ASCII and EBCDIC.

If you decide to use a different character set, then you need to set NLS_LANG accordingly. For example, assuming you need to use Swedish characters on OS/390 UNIX System Services, the following might set appropriate values into NLS_LANG:

```
export NLS_LANG=Swedish_Sweden.S8EBCDIC278
```

LIBPATH
The LIBPATH environment variable controls the search path for DLLs or shared objects under OS/390 UNIX System Services. Several Oracle executables use DLLs. To locate the Oracle DLLs, LIBPATH must include the `$ORACLE_HOME/lib` directory. You can prepend the `$ORACLE_HOME/lib` directory to the LIBPATH environment variable by issuing the following command:

```
export LIBPATH=$ORACLE_HOME/lib:$LIBPATH
```

PATH
The PATH environment variable controls the search path for OS/390 UNIX System Services programs. It is a list of directories in the following form:

```
absolute_pathname[:absolute_pathname ...]
```

This chapter assumes that the path to the Oracle9*i* program has been specified in the PATH environment variable. For example:

```
export PATH=$PATH:$ORACLE_HOME/bin
```

File Names in OS/390 UNIX System Services

The Oracle OS/390 utilities support a number of file name features that are unnecessary on OS/390 UNIX System Services.

- DD statements are not normally used in OS/390 UNIX System Services, so the `/DD/ddname` form of naming a file is not supported.
- File names refer to files in the HFS and adhere to POSIX naming rules. Files designated by the /DSN/ method of naming files are in the OS/390 file system, not the HFS. The /DSN/ notation is not supported.
- On OS/390, the Oracle FNA service supports the construction of file names compatible with the OS/390 file system. Files in the HFS are accessed, not OS/390 files, so FNA is not supported.
- Parameter redirection (`++/DD/ddname` or `++/DSN/dsname`) is not supported. In addition to the use of OS/390 file system files, the facility is designed for the OS/390 JCL PARM field; OS/390 UNIX System Services does not have access to the PARM field.

Accessing OS/390 Data Sets

When run under USS, the `imp`, `exp`, and `sqlldr` utilities are called in the same manner as on other UNIX platforms. In addition, these utilities can access OS/390

data sets by simply specifying the filenames by preceding them with a double forward slash and enclosing them in single quotes.

Example 4–1 //'ORACLE.EXPEMPL'

```
//'ORACLE.EXPEMPL'
```

If you specify the filename on the command line or at a utility prompt, you must escape all forward slashes and the single quotes.

Example 4–2 \/\/\'ORACLE.EXPEMP\'

```
\/\/\'ORACLE.EXPEMP\'
```

The following is an example of running `exp` under UNIX System Services:

```
exp

Export: Release 9.0.1.0.1 - Production on Thu Aug 10 10:53:20 2000

(c) Copyright 2000 Oracle Corporation.  All rights reserved.


Username: scott
Password:
Connected to: Oracle9i Enterprise Edition Release 9.0.1.0.1 - Production
With the Partitioning option
Enter array fetch buffer size: 4096 >

Export file: expdat.dmp > \/\/\'ORACLE.EXPEMP\'

(2)U(sers), or (3)T(ables): (2)U > 3

Export table data (yes/no): yes >

Compress extents (yes/no): yes >

Export done in WE8EBCDIC1047 character set and WE8EBCDIC1047 NCHAR character set

About to export specified tables via Conventional Path ...
Table(T) or Partition(T:P) to be exported: (RETURN to quit) > EMP
. . exporting table                            EMP         14 rows exported
Table(T) or Partition(T:P) to be exported: (RETURN to quit) >
Export terminated successfully without warnings.
```

Utilities Available Under OS/390 USS

The following utilities are available under OS/390 UNIX System Services.

PL/SQL Wrapper

The PL/SQL wrapper enables you to produce a binary version of your PL/SQL packages that is suitable for shipping in a tamper-proof way. To run it, enter the following at the command prompt:

```
wrap
```

For additional information, refer to the *PL/SQL User's Guide and Reference*.

PL/SQL Server Page Loader

The Oracle9*i* PL/SQL Server Page Loader enables you to load PL/SQL Server Pages (PSP) into the database as stored procedures.

For example:

```
loadpsp -replace -user scott/tiger@WEBDB banner.inc error.psp display_order.psp
```

Additional information can be found in the *Oracle9i Application Developer's Guide - Fundamentals*, in the chapter named "Developing Web Applications with PL/SQL".

Data Guard Command-line Utility

The Data Guard command-line interface allows you to control and monitor a Data Guard configuration from the DGMGRL command-line prompt or from within scripts. You can perform most of the activities required to manage and monitor the objects in the configuration using the command-line interface.

To run the Data Guard command-line interface, you must have SYSDBA privileges. Start the command-line interface by entering DGMGRL at the command line prompt on a system where Oracle9*i* Data Guard is installed:

```
$ DGMGRL [options]
```

Additional information can be found in the *Oracle9i Data Guard Concepts and Administration Guide*.

OEM Intelligent Agent and Data Gatherer

OEM Intelligent Agent allows you to monitor events, start and stop an Oracle Instance in OS/390. Data Gatherer allows you to gather database and operating system statistics on OS/390. See the *Oracle9i Enterprise Edition System Administration Guide for OS/390* and *Oracle Intelligent Agent User's Guide* for further information.

Oracle JDBC Thin Driver

The Oracle JDBC thin driver is a Type IV JDBC driver targeted to application developers. Written entirely in Java, it complies fully with the JDBC 1.2.2 standard. It requires the use of TCP/IP. For additional information, refer to the *Oracle9i JDBC Developer's Guide and Reference*.

To run the Oracle JDBC thin driver, you must add the following to the CLASSPATH environment variable that is in effect when the JDBC driver runs:

```
$ORACLE_HOME/jdbc/lib/classes111.zip      /*Oracle JDBC thin driver*/
<installation-specific path>              /*IBM JDK classes        */
```

SQLJ Translator

The SQLJ translator is conceptually similar to other Oracle Precompilers, SQLJ consists of both a translator and a runtime component. The translator replaces embedded SQL calls to the SLQJ runtime, which implements the SQL operations. When the end user runs the SQLJ application, the runtime is invoked to handle the SQL operations in real-time. For additional information, refer to the *Oracle9i JDBC Developer's Guide and Reference*.

To run the SQLJ translator, you must add the following to the CLASSPATH environment variable that is in effect when SQLJ runs. For example:

```
current directory
$ORACLE_HOME/sqlj/lib/translator.zip      /*SQLJ translator        */
$ORACLE_HOME/jdbc/lib/classes111.zip      /*Oracle JDBC thin driver*/
<installation-specific_path>              /*IBM JDK classes         */
```

Loadjava/Dropjava Utilities

The `loadjava` utility is used to load java source code or byte code into an Oracle database as Java stored procedure. The `dropjava` removes it.

For example:

```
> loadjava  -thin -verbose -user scott/tiger@144.25.40.59:1493:ORAJ
-resolve  Test1.class
initialization complete
loading  : Test1
creating : Test1
resolver :
resolving: Test1
```

A sample session for `dropjava` of a class file:

```
> dropjava -thin -verbose -user scott/tiger@144.25.40.59:1493:ORAJ
Test1.class
dropping class   : Test1
```

A sample session for `loadjava` of a source file:

```
> loadjava -thin -verbose -user scott/tiger@144.25.40.59:1493:ORAJ
-resolve -encoding Cp1047  Test2.java
initialization complete
loading  : Test2
creating : Test2
resolver :
resolving: Test2
```

A sample session for `dropjava` of a source file:

```
> dropjava -thin -verbose -user scott/tiger@144.25.40.59:1493:ORAJ
Test2.java
dropping source  : Test2
```

Oracle Wallet Manager

Oracle Wallet Manager (OWM) is a Java application that requires Java 1.1.8, an X11 server and a graphics terminal. OWM allows the user to administer certificates in his wallet directory, including generating certificate requests, importing/exporting user certificates and importing/exporting trustpoint certificates. See the *Oracle Label Security Administrator's Guide* for more information.

A sample session for OWM could be started as follows:

```
export DISPLAY=my_workstation:0.0
owm
```

where: `my_workstation` is your workstation name or IP address and the environment variable ORACLE_HOME must be set.

If Java returns the following message, your DISPLAY value is invalid:

```
java.lang.LNullPointerException
java.lang.NullPointerException
        at oracle.ewt.lwAWT.BufferedFrame._init(Unknown Source)
        at oracle.ewt.lwAWT.BufferedFrame.<init>(Unknown Source)
        at

oracle.sysman.emSDK.client.appContainer.ApplicationFrame.<init>(ApplicationFrame
.java:75)

        at

oracle.sysman.emSDK.client.appContainer.WebApplication.main(WebApplication.java:
2908)
```

TNSPING

TNSPING is a command line executable that tests tns connectivity. See the *Oracle9i Net Services Administrator's Guide* for more information.

Character Set Scanner

Refer to Chapter 10 of the *Oracle9i Database Globalization Support Guide* for information on using the Character Set Scanner utility. Also refer to Appendix D of the *Oracle9i Enterprise Edition System Administration Guide for OS/390* to obtain the list of supported character sets for OS/390 databases.

Please note that the Character Set Scanner under UNIX System Services can only support databases running with the EBCDIC character set.

Locale Builder

To run the Locale Builder, you must have the `JAVA_HOME` environment variable set to the root directory that JAVA is currently installed under.

For example:

```
export JAVA_HOME=/usr/lpp/java/J1.1
```

Also, you may have to customize the `JAVALIB` and `CLASSPATH` environment variable in shell script `LBuilder.ksh` to point to the current Oracle, Sun, and IBM classes.

For example:

```
JAVALIB
```

```
  classes.zip                          for JRE 1.1.8
CLASSPATH
  current directory
  LocaleBuilder.jar
  swingall.jar                         required for JRE 1.1.8 only
  jewt-all-dbg-4_1_0.jar               required for JRE 1.1.8 only
```

Please note that the Locale Builder is currently supported under JRE 1.1.8 only.

Customer-Written Applications Under USS

Oracle9*i* database servers can be accessed by programs run from the OS/390 UNIX System Services Hierarchical File System (HFS). These programs must be coded in C programming language and must utilize OCI or the Oracle Pro*C Precompiler. You need a thorough understanding of the following:

- OS/390 UNIX System Services environment
- IBM C/370 or IBM C/C++ compiler
- DFSMS/MVS Program Management binder
- IBM prelinker
- LE/370 runtime library
- OS/390 UNIX System Services `c89` shell command

OCI programs can be compiled, prelinked, and linkedited (bound) entirely within the OS/390 UNIX System Services shell using the `c89` command. Oracle Pro*C precompiler programs can now be precompiled, compiled, prelinked, and linkedited (bound) entirely within the OS/390 UNIX System Services shell. An OS/390 UNIX System Services compatible version of the IBM C/370 or IBM C/C++ compiler and LE/370 runtime library must be used. Depending on the nature of the application, you might also need to call the C/370 or C/C++ prelinker.

Once the OCI or Oracle Precompiler application is created, it is typically run from the OS/390 UNIX System Services shell. The environment variables described in "Running an Oracle Utility on OS/390 Under USS" on page 4-4 are also applicable for the execution of user-written programs.

Child Process Restrictions

An application using a cross-memory connection to Oracle (PROTOCOL=XM) cannot use the `fork()` system call to create a child process. This is because the

cross-memory protocol adapter uses a thread to field the SIGINT signal from the Ctrl-C key while it is in cross-memory mode. The UNIX System Services kernel will not deliver a signal to a program running in cross-memory mode. Applications should use the `spawn()` system call instead. If the application must use `fork()`, it must use TCP/IP to connect to the database.

The Oracle9*i* database server does not allow a child process to make use of a connection established by a parent. If this is attempted, then the result is:

```
CEE3250C The system or user abend S0D6  R=00000022 was issued.
```

The connect capabilities for a child process after a fork are:

POSIX Thread Support

OS/390 UNIX System Services also provides for POSIX threading. If the application uses threads, then it must ensure two or more threads do not attempt to use the same connection simultaneously.

5

Export and Import Utilities

The Export and Import utilities are used to move Oracle database tables and other objects from one database to another. The databases can be on the same platform (for instance, both OS/390) or on different platforms.

The executables for Oracle Export and Import are named EXP and IMP. If you make these available to your TSO session by placing them in your STEPLIB, it disables the abbreviated forms of the native (IDCAMS) export and import commands, which are also EXP and IMP.

This chapter describes the function of each of these utilities as they relate to the OS/390 operating system and contains the following sections:

The information in this chapter supplements the documentation for the Oracle utilities in *Oracle9i Database Utilities*.

Export

The Export (EXP) utility reads data from the Oracle9*i* database according to your request and writes an OS/390 sequential data set. Export is used to provide backups of the Oracle tables and database. It is also used to move data from one Oracle9*i* database to another.

The Export utility has additional functions for users with Oracle DBA authority. For example, to perform incremental or cumulative Exports, DBA authority is required. Refer to the *Oracle9i Enterprise Edition System Administration Guide for OS/390* for a description of the Export utility for database administrators.

Running Under UNIX System Services

When running Export under USS, considerations are the same as described in the *Oracle9i Database Utilities* manual. Refer to Chapter 4, "Accessing Oracle9i Under USS", for general information about running utilities in the USS environment.

Running Under TSO

The syntax for running Export under TSO is:

```
EXP [userid[/password[@connect-string]]
```

where:

`userid`	is the Oracle user id. This parameter is optional.
`password`	is the password associated with `userid`. This parameter is optional.
`connect-string`	is a tnsnames alias name or a complete Oracle Net address string. This parameter is optional if the Oracle database server is specified using methods described in Chapter 2. Refer to Chapter 9 for further details.

The normal OS/390 EOF key sequence, `/*`, ends the Export utility from any prompt. The alias of EXP is ORAEXP.

Running in Batch

Export is supported in the batch environment through the ORAEXP JCL procedure. The following is a copy of this procedure:

```
//ORAEXP   PROC INDEX=oran,
//              LIBV=orav,
//              SYSOUT='SYSOUT=*',
//              USERID='NAME/PASSWORD'
//ORAEXP   EXEC PGM=EXP,REGION=4M,
//              PARM='&USERID'
//STEPLIB  DD   DSN=&INDEX..&LIBV..CMDLOAD,DISP=SHR
//ORA$LIB  DD   DSN=&INDEX..&LIBV..MESG,DISP=SHR
//SYSOUT   DD   &SYSOUT,DCB=(LRECL=132,BLKSIZE=1320,RECFM=VB)
//SYSERR   DD   SYSOUT=*,DCB=(LRECL=132,BLKSIZE=1320,RECFM=VB)
//ORAPRINT DD   SYSOUT=*
```

To run this JCL procedure, you must include additional DD statements that refer to other required data sets. The following example assumes the Oracle9*i* database server is installed with a system identification (SID) of ORA1:

```
//USER1JOB JOB  ........
//STEP1    EXEC ORAEXP,
// PARM='SCOTT/TIGER PARFILE=/DD/PARFILE'
//ORA@ORA1 DD   DUMMY
//SYSIN    DD   DUMMY
//EXPTFL   DD   DSN=userid.EXPDTFL.TEST,
//              DISP=(NEW,CATLG,DELETE),
//              VOL=SER=TEMP01,UNIT=SYSDA,
//              SPACE=(TRK,(5,5),RLSE),
//              DCB=(RECFM=FB,LRECL=4096,BLKSIZE=24576)
//PARFILE  DD   *
DIRECT=N
BUFFER=10240
FILE=/DD/EXPTFL
GRANTS=Y
INDEXES=Y
ROWS=Y
CONSTRAINTS=Y
COMPRESS=N
FULL=N
TABLES=(DEPT)
/*
```

In this example, Oracle user SCOTT with password TIGER performs an Export of data from the database instance whose OSDI service SID ORA1. The Export parameters reside in PARFILE instream data. As an alternative, the Export responses can be included as SYSIN instream data.

To further illustrate, you can use Export in two ways:

- Enter the command EXP user name and password PARFILE=`filename` with the remaining Export parameters contained in the specified file. This is the preferred method. If you use this method, then PARFILE cannot be a PDS member.
- Enter the command EXP user name and password followed by various parameters that control how Export runs. You can use this method as long as the number of parameters does not exceed the maximum length of a command line on your system. The parameter redirection attribute ++ can be used to redirect parameters to the input file.

For more information on using Export, refer to *Oracle9i Database Utilities*.

Return Codes

Export issues the following return codes:

0 is a normal (successful) completion. Review results carefully as a 0 return code can indicate an error not serious enough to terminate the utility, but invalidates the Export file.

8 is an end due to an irrecoverable error. The error can be one of the following:

- File error (in this case Export ends with an X'37' abend).
- Insufficient parameters in the input file when Export is run in batch.
- Export session ended with a period (`.`) instead of a null entry. The Export log file contains an error message describing the error.

Exporting to Non-OS/390 Systems

When exporting to non-OS/390 systems, ensure the data transfer process does not result in the translation of the Export file from EBCDIC to ASCII or from one EBCDIC character set to another. This translation, if necessary, is provided by the Import utility on the receiving system. If an external translation is performed, then Import ends with an irrecoverable error and you receive the following error message:

```
Seals don't match
```

You must also ensure Exports to other operating system environments are done on compatible media.

Import

The Import (IMP) utility reads data from a sequential data set prepared by the Export utility and writes data into an Oracle9*i* database.

The Import utility has additional functions for users with Oracle DBA authority. Refer to the *Oracle9i Enterprise Edition System Administration Guide for OS/390* for a description of Import with DBA authority.

Running Under UNIX System Services

When running Import under USS, considerations are the same as described in the *Oracle9i Database Utilities* manual. Refer to Chapter 4, "Accessing Oracle9i Under USS", for general information about running utilities in the USS environment.

Running Under TSO

The syntax for running the Import utility under TSO is:

```
IMP [userid[/password[@connect-string]]
```

where:

`userid`	is the Oracle user id. This parameter is optional.
`password`	is the password associated with `userid`. This parameter is optional.
`connect-string`	is a tnsnames alias name or a complete Oracle Net address string. This parameter is optional if the Oracle database server is specified using methods described in Chapter 2. Refer to Chapter 9 for further details.

Use the normal OS/390 EOF key sequence (`/*`) to end the Import utility from any prompt. The IMP alias is ORAIMP.

Running in Batch

Import is supported in the batch environment through the ORAIMP JCL procedure. A copy of this procedure is:

```
//ORAIMP   PROC INDEX=oran,
//              LIBV=orav,
//              SYSOUT='SYSOUT=*',
//              USERID='NAME/PASSWORD'
//ORAIMP   EXEC PGM=IMP,REGION=4M,
//              PARM='&USERID'
//STEPLIB  DD   DSN=&INDEX..&LIBV..CMDLOAD,DISP=SHR
//ORA$LIB  DD   DSN=&INDEX..&LIBV..MESG,DISP=SHR
//SYSOUT   DD   &SYSOUT,DCB=(LRECL=132,BLKSIZE=1320,RECFM=VB)
//SYSERR   DD   SYSOUT=*,DCB=(LRECL=132,BLKSIZE=1320,RECFM=VB)
//ORAPRINT DD   SYSOUT=*
```

The following JCL illustrates the use of the previous procedure:

```
//USER1JOB JOB  ........
//STEP1    EXEC ORAIMP,
//              USERID=,
//              PARM='PARFILE=/DD/PARFILE'
//ORA@ORA1 DD   DUMMY
//SYSIN    DD   DUMMY
//EXPTFL   DD   DSN=userid.EXPDTFL.TEST,DISP=SHR
//PARFILE  DD   *
USERID=SCOTT/TIGER
BUFFER=10240
FILE=/DD/EXPTFL
SHOW=N
IGNORE=N
GRANTS=Y
INDEXES=Y
ROWS=Y
FULL=Y
COMMIT=Y
/*
```

In this example, data is imported to the database instance whose OSDI service SID is ORA1. The Import parameters reside in PARFILE instream data. These responses can alternatively be specified as SYSIN instream data.

To further illustrate, you can use Import in two ways:

- Enter the command IMP user name and password PARFILE=`filename` with the remaining Import parameters contained in the specified file. This is the preferred method. If you use this method, then PARFILE cannot be a PDS member.
- Enter the command IMP user name and password followed by various parameters that control how Import runs. You can use this method as long as the number of parameters does not exceed the maximum length of a command line on your system. The parameter redirection attribute ++ can be used to redirect parameters to the input file.

For more information about Import, refer to *Oracle9i Database Utilities.*

Return Codes

Import provides one of the following return codes:

0 is a normal (successful) completion. Review results carefully as a 0 return code can indicate an error not serious enough to end the utility, but invalidates the Import file.

8 is an end due to an irrecoverable error. The Import log file contains an error message describing the error.

Importing from Non-OS/390 Systems

When importing from non-OS/390 systems, ensure your data transfer process does not translate the Export file from ASCII to EBCDIC. This translation, if needed, is provided by the Import utility on the OS/390 system. If an external translation is performed, then Import ends with an irrecoverable error and you receive the following error message:

```
Seals don't match
```

You must also ensure Exports done for eventual Import to OS/390 environments are done on compatible media.

Note: Importing partitioned tables that are partitioned on character values into an OS/390 table might not be possible if they were exported from an ASCII database. The same is true for importing into an ASCII database after exporting from OS/390. If this is required, then use conventional path export and create the table before importing.

6

SQL*Loader

SQL*Loader is a utility that loads from sequential files into tables in an Oracle9*i* database.

Read this chapter to understand how this powerful utility's advanced function data loader capabilities are used under OS/390.

The following sections are discussed in this chapter:

This information supplements the documentation for SQL*Loader in *Oracle9i Database Utilities.*

Running Under UNIX System Services

When running SQL*Loader under USS, considerations are the same as described in the *Oracle9i Database Utilities* manual. Refer to the section "OS/390 UNIX System Services Overview" on page 4-1 for general information about running utilities in the USS environment.

Running Under TSO

Use SQL*Loader to load data from sequential files into the Oracle9*i* database. Use a control file to describe the data location, data format, and target tables. You can direct data that does not satisfy specified conditions to special files.

The following is the syntax of the SQL*Loader command under TSO:

```
SQLLDR keyword=value [keyword=value,...]
```

The parameters passed to SQL*Loader are keyword parameters. The parameters are fully described in *Oracle9i Database Utilities.*

SQL*Loader Files

SQL*Loader uses the following files:

- A BAD file to store records from the input data file that fail basic validation checks, such as nonnumeric data in a numeric field
- A CTL file containing all the control information
- A DAT file containing the data to be loaded
- A DSC file to store the discarded records from the input data file, which are records that do not qualify to be loaded into the database
- A LOG file containing entries describing what SQL*Loader did

SQL*Loader File Names

The section "File Name/Attribute Augmentation (FNA) Facility" on page 3-8 describes extension and FNA processing that each of the file names supplied to SQL*Loader is subject to. The filetype extensions are BAD, CTL, DAT, DSC, and LOG.

For example, without any special FNA controls, the following specification results in SQL*Loader using `tsoprefix`.TESTDATA.CTL as the control data set:

```
SQLLDR ... CONTROL=TESTDATA
```

In batch or when using TSO without a PROFILE PREFIX, TESTDATA.CTL is used as the control file. This data set is not a good choice in most OS/390 systems because TESTDATA is not a proper high-level qualifier for a data set name.

If any file other than the control file is not specified, then SQL*Loader uses the control file name to derive the name of the file. For example, if the LOG parameter is omitted but CONTROL=TEST3 is specified, then a log file name of `tsoprefix`.TEST3.LOG is used. The same processing applies to the BAD, DAT, and DSC files if any of these are not specified.

To have a BAD file generated, you must specify a BAD filetype.

If the supplied control file name is one that cannot be modified with an extension (that is, it is prefixed with a pathname surrounded by slashes), then the name cannot be used to derive the names of any omitted files. In this case, SQL*Loader uses the default name LOADER for the derivation.

For example, to call SQL*Loader use the following command:

```
SQLLDR USERID=SCOTT/TIGER CONTROL=/DD/CTLFILE ...
```

This command results in the derived LOG file name `tsoprefix`.LOADER.LOG because /DD/CTLFILE cannot be used to derive the other distinct file names.

This approach to deriving file names can be convenient in the TSO environment. But when running in batch where there is no TSO PROFILE PREFIX it might be difficult to use, depending on the data set naming standards and allocation controls at your installation. The best approach to running SQL*Loader in batch is to specify all required files with /DSN/ or /DD/ notation:

```
//LOAD EXEC PGM=SQLLDR,
// PARM='CONTROL=/DD/CTL DATA=/DD/DATA LOG=/DD/LOG BAD=/DD/BAD DISCARD=/DD/DIS'
//CTL  DD  ...
//DATA DD  ...
//LOG  DD  ...
//BAD  DD  ...
//DIS  DD  ...
```

SQL*Loader File Attributes

SQL*Loader for OS/390 differs from SQL*Loader for other systems in the way file attributes are decided for the BAD and DSC files. In other systems, the file attributes for these files are made identical to the data file. This is done so you can make corrections to these files and recycle them through SQL*Loader again. In OS/390, these data sets are created with default DCB attributes that are generally different from the attributes of the input file. There are two ways to overcome this:

1. You can, in your JCL or TSO ALLOC command for the BAD/DSC file, specify the DCB attributes of the BAD/DSC files be the same as that of the data file by a refer back, as shown in the following example:

```
//BAD DD DISP=SHR,DSN=PROD.CASE1.BAD,DCB=PROD.CASE1.DAT
//DIS DD DISP=SHR,DSN=PROD.CASE1.DSC,DCB=PROD.CASE1.DAT
```

2. For each invocation of SQL*Loader you can have a different FNA control file allocated to the ORA$FNA DD statement at runtime. This FNA control file can contain FSA entries for suffixes BAD and DSC specifying the correct FATTR parameters.

Running in Batch

SQL*Loader is supported in the batch environment through the ORALDR JCL procedure. A copy of this procedure is reproduced in the following example. The example assumes the Oracle9*i* database server has been installed with the system identification (SID) of ORA1.

```
//ORALDR PROC INDEX='oran',
//*                                        NONVSAM LIBRARY HLINDEX.
//             LIBV='orav',
//*                                  ORACLE/VERSION AND INSTALL LEVEL.
//             INDD=LDRCTL,           INPUT CONTROL FILE DDNAME.
//             SYSOUT='SYSOUT=*',     SYSOUT CLASS.
//             LOGDD=LOGDD,           OUTPUT REPORT.
//             USERID='NAME/PASSWORD', USERID
//             D=FALSE                DIRECT=TRUE FOR FAST LOADER
//*
//*                                                                    *
//*                  SQL*LOADER BATCH PROCESSOR                        *
//*
//ORALDR   EXEC PGM=SQLLDR,REGION=4M,
// PARM=('CONTROL=/DD/&INDD USERID=&USERID',
//       'LOG=/DD/&LOGDD DIRECT=&D')
//STEPLIB  DD DSN=&INDEX..&LIBV..CMDLOAD,DISP=SHR
//ORA$LIB  DD DSN=&INDEX..&LIBV..MESG,DISP=SHR
//SYSOUT   DD &SYSOUT,DCB=(LRECL=132,BLKSIZE=1320,RECFM=VB)
//SYSERR   DD SYSOUT=*,DCB=(LRECL=132,BLKSIZE=1320,RECFM=VB)
//ORAPRINT DD SYSOUT=*
//SYSIN    DD DUMMY
//*
//*  REQUIRES //ORA@SID  DD DUMMY STATEMENT (ORACLE SUBSYSTEM NAME).
//*  ALSO REQUIRES BADDD AND DSCDD TO SPECIFY BAD AND DSC FILES
```

```
//*
```

When running this procedure, you must include additional DD statements. These DD statements refer to other required data sets.

The following example illustrates the use of the previous procedure:

```
//USER1JOB JOB  ..............
//S1  EXEC ORALDR,
// PARM='CONTROL=/DD/CTL USERID=A/B BAD=/DD/BAD DATA=/DD/DAT
//        DISCARD=/DD/DIS'
//CTL  DD DISP=SHR,DSN=PROD.CASE1.CTL
//DAT  DD DISP=SHR,DSN=PROD.CASE1.DAT
//BAD  DD DISP=SHR,DSN=PROD.CASE1.BAD,DCB=PROD.CASE1.DAT
//DIS  DD DISP=SHR,DSN=PROD.CASE1.DSC,DCB=PROD.CASE1.DAT
//SYSIN  DD DUMMY
//ORA@ORA1 DD DUMMY
```

When called in batch, the command line arguments are often longer than is allowed in the PARM field of the EXEC statement. To work around this, use the standard Oracle parameter redirection operation as in any other Oracle utility. Refer to "Redirecting Standard Files and Parameters" on page 3-6 for more information.

Return Codes

SQL*Loader issues the following return codes:

0 is a normal (successful) completion.

4 indicates a discontinued load. This is due to a runtime error, such as running out of file extents.

8 is an end due to an irrecoverable error.

SQL*Loader VSAM File Support

SQL*Loader for OS/390 is enhanced to provide native VSAM support. This feature lets you load Oracle tables directly from VSAM key sequenced data set (KSDS), entry sequenced data set (ESDS), and relative record data set (RRDS) clusters. All the normal SQL*Loader operating capabilities, such as loading multiple tables and discard logic, work with VSAM clusters in the same way as with non-VSAM data sets.

Specifying VSAM Input to SQL*Loader

The file name supplied with the INFILE or INDDN keyword can take one of three forms:

/DD/name	is the one-character to eight-character name of a DD statement or TSO allocation already present in the jobstep or TSO session. The DD statement or allocation specifies the VSAM cluster to be loaded.
/DSN/name	is the 1-character to 44-character, fully-qualified data set name of the VSAM cluster. SQL*Loader dynamically allocates the cluster with DISP set to SHR before opening it.
name	a file name that does not begin with /DD/ or /DSN/ is assumed to be an unqualified data set name. In this case, SQL*Loader appends a TSO user's PROFILE PREFIX on the left side of name and proceeds with dynamic allocation and open as if /DSN/ is used. When no PROFILE PREFIX is defined for a TSO user and when running in batch, name is processed as a fully-qualified name just as though a /DSN/ pathname prefix is included.

In all these cases, name can optionally include a slash followed by a VSAM cluster password on the right side. Because SQL*Loader only opens VSAM clusters for input, a cluster's READPW (read password) or any higher-level password can be specified. If a cluster does not have a read password, then no password must be specified. If a cluster has a read password and the correct password is not supplied in the file name, then VSAM performs normal prompting of the TSO user or system operator when SQL*Loader opens the cluster. VSAM password prompting is affected by the ATTEMPTS parameter of the IDCAMS DEFINE or ALTER command.

Examples

The following partial examples of SQL*Loader LOAD DATA statements show how to specify VSAM files in various ways.

The VSAM cluster GJONES.PAYSEQ2.MASTER is dynamically allocated for input; the password ZAP is included:

```
LOAD DATA INFILE /DSN/GJONES.PAYSEQ2.MASTER/ZAP ...
```

Assuming a TSO user with a PROFILE PREFIX of GJONES, the VSAM cluster GJONES.MY.RRDS is dynamically allocated for input:

```
LOAD DATA INFILE MY.RRDS ...
```

A password is also supplied:

```
LOAD INFILE /DD/VSAMDD/password ...
```

For the following specification, the job includes a DD statement:

```
//VSAMDD DD DISP=SHR,DSN=GJONES.RECVBL.LOG
```

Or, if it is being used in a TSO session, a prior TSO ALLOC command can be issued:

```
ALLOC FILE(VSAMDD) DA('GJONES.RECVBL.LOG') SHR
```

SQL*Loader VSAM Processing Considerations

SQL*Loader VSAM processing is generally the same as for non-VSAM input files. The following considerations are unique to VSAM:

- SQL*Loader reads KSDS clusters in key sequence, RRDS clusters in record number sequence, and ESDS clusters in RBA sequence. Empty record slots in an RRDS are not read and do not contribute to SQL*Loader SKIP or LOAD counts.
- SQL*Loader also processes an Alternate Index Path (AIX) as input. Oracle Corporation recommends loading directly from the base cluster since AIX only changes the order in which records are read and can significantly reduce performance.
- All VSAM-specific error or warning messages issued by SQL*Loader are written to the C programming language standard error file (normally the SYSERR DD allocation in batch or the user's screen in TSO).
- If SQL*Loader is unable to dynamically allocate a VSAM cluster, then the return, error, and information codes from OS/390 dynamic allocation are reported and the load ends.
- If the OPEN of a VSAM cluster fails, then the return code and ACBERFL (ACB error code) value are reported and the load ends.
- Warning conditions indicated by VSAM OPEN are reported by SQL*Loader but do not end processing.
- VSAM clusters containing invalid control intervals (that is, lacking valid CIDF/RDF fields) cannot be processed by SQL*Loader.

- If a VSAM GET request fails, then the return, component, and RPL feedback codes are reported and the load ends.
- If the CLOSE of a VSAM cluster fails, then the return code and ACBERFL value are reported, but the already completed load is unaffected.

Return and related codes from OS/390 dynamic allocation and VSAM request macroinstructions are documented in the appropriate IBM documentation.

BAD and DISCARD File Considerations with VSAM Input

When an input record to SQL*Loader encounters an Oracle error during insertion or is not selected for insertion due to WHEN criteria, it is written to the BAD or DSC file, respectively. These files are normally identical to the input data file in structure and format.

When a VSAM file is used as input to SQL*Loader, the BAD and DSC files remain non-VSAM in structure (as they are with non-VSAM input). This eliminates the need to DEFINE additional VSAM files each time SQL*Loader is used with VSAM input.

With VSAM input, SQL*Loader normally requires the BAD and DSC files to use RECFM=V or VB and LRECL=n, where n is greater than or equal to the maximum record length of the input VSAM file. When the input VSAM file is known to have all records the same length (such as in an RRDS), you can use RECFM=F or FB and LRECL=x, where x is the exact length of the input VSAM records. Use caution when using this format for non-RRDS clusters, however, because VSAM has no mechanism to ensure all records in a KSDS cluster or ESDS cluster are actually the same size.

SQL*Loader Direct Path

The SQL*Loader direct path (DIRECT option) has significant performance improvements for many data load applications. Some of the performance improvements include reduced CPU usage and reduced time to perform a data load. Refer to *Oracle9i Database Utilities* for a complete description of the SQL*Loader direct path.

Performance

When you run SQL*Loader with the DIRECT option set to TRUE, performance is significantly influenced by the number of QSAM buffers available for processing the input file. The number of buffers is controlled by the DCB BUFNO parameter

value you specify when you allocate the data file. The default value is five if you do not specify a BUFNO value. You can dramatically increase the data load rate in DIRECT mode by increasing the number of buffers to 100 or more.

Tests show that 200 buffers can increase the data load rate by a factor of three or more over the data load rate with the default number of buffers. SQL*Loader performance does not appear to improve with more than 200 buffers allocated for the input file.

Increase the number of buffers by providing a BUFNO value when the input file is allocated. For example, if you are running SQL*Loader as a batch job, then provide 100 buffers for the input file by allocating the file with the DCB=BUFNO=100 parameter as shown in the following example:

```
//DATA  DD  DSN=input.file.name,DISP=SHR,DCB=BUFNO=100
```

If you are running SQL*Loader from TSO, then provide 100 buffers for the input file by adding the BUFNO parameter to the ALLOC command that allocates the file. For example:

```
ALLOC FILE(DATA) DA('input.file.name') SHR BUFNO(100)
```

Ensure the region available to SQL*Loader includes enough memory to hold the extra buffers. For example, a file with a blocksize of 23,476 allocated with 100 input buffers requires 2,347,600 bytes of memory for the buffers. You might need to increase the region parameter value to 4M or more to run SQL*Loader with many buffers.

With 200 input file buffers available, SQL*Loader running with the DIRECT set to the TRUE option can use more than 90% of one central processor, and attain data load rates of over 250K per second on some types of processors. If you do not want SQL*Loader to use this much CPU, then reduce the data load rate by reducing the BUFNO value for the input file to 20 or less.

Each SQL*Loader session runs as a single OS/390 task and does not use more than one central processor in a multiprocessor complex. For example, the maximum amount of CPU a single SQL*Loader session can use in a four-way processor complex is 25 percent.

To exploit more than one processor in a multiprocessor complex, use multiple SQL*Loader sessions as documented in *Oracle9i Database Utilities*.

7

Oracle Precompilers

Oracle Precompilers are used to process C, COBOL, or PL/I programs before passing them to their native compilers. Oracle Precompilers translate embedded SQL statements into calls necessary to access the Oracle9*i* database server.

Read this chapter for OS/390-specific information about using the Oracle Precompilers and Chapter 8, "Oracle Call Interface" for information about the Oracle Call Interface. Sections in this chapter include:

The product-specific documentation for the Oracle Precompilers is contained in the following Oracle documents:

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

Oracle Precompilers Overview

The Oracle Precompilers are used to process C, COBOL, or PL/I programs before passing them to their native compilers. Oracle Precompilers translate embedded SQL statements in the programs into the appropriate native language statements and calls necessary to access the Oracle9*i* database server.

OS/390 supports several major operating environments, and the Oracle Precompilers can be used to target your programs to run in any of them. Whether your application requires traditional batch processing, interactive processing under TSO, or transaction processing under CICS or IMS TM, the Oracle Precompilers are the first step in building your programs.

Oracle also supports the OS/390 UNIX System Services (USS) environment. Refer to Chapter 4, "Accessing Oracle9i Under USS", for additional information about building and running Oracle Precompiler applications in the OS/390 UNIX System Services environment.

Regardless of the environment you choose, there are four steps involved:

1. Precompiling your program with one of the Oracle Precompilers and other precompilers that are required by the target environment.
2. Compiling the program with the language's compiler.

> **Note:** A Language Environment conforming compiler is required for programs targeted to run in BATCH or TSO.

3. Linking the program with an Oracle interface stub designed especially for the target environment. Batch, TSO, CICS, and IMS TM each require linking with a different stub.
4. Running your program in the target environment.

This chapter discusses each of the steps needed to build and run an Oracle Precompiler application program, and contains some environment-specific program design considerations. For more information, refer to *Oracle9i Application Developer's Guide - Fundamentals* and the precompiler documentation specific to the programming language you plan to use.

Target Environment Design Considerations

The following information discusses application design considerations that apply to each of the target environments.

TSO Programs

To use an Oracle Precompiler application under TSO (by the CALL command or as a command processor), ensure an EXEC SQL COMMIT WORK RELEASE or EXEC SQL ROLLBACK WORK RELEASE statement is run before the program ends

The IBM TSO TEST processor does not support cross memory mode operations. Because Oracle applications perform all database operations in cross memory mode, you cannot escape to TSO TEST when an Oracle application is performing operations within the database.

CICS Programs

Oracle Access Manager for CICS allows COBOL or C programming language programs running as CICS transactions to access an Oracle9*i* database server anywhere in your network. Programs can access and update DL/I, VSAM, and Oracle9*i* data within a single synchronized transaction.

From a CICS perspective, programs accessing Oracle9*i* databases have no special requirements. The normal CICS SEND and RECEIVE functions are used to communicate with the CICS terminal.

From an Oracle9*i* perspective, Oracle Precompiler applications running under CICS are no different from other Oracle Precompiler applications. They contain SQL statements that are translated by Oracle Precompilers and passed to the Oracle9*i* database server as the program runs. When using Pro*C or Pro*COBOL programs in a CICS transaction, however, there are some special considerations.

Use of CONNECT...AT

Oracle9*i* programs often use the AT clause of the CONNECT statement to specify which server to access.

Synchronization of Oracle and CICS Updates

Your CICS programs can, optionally, synchronize Oracle updates with updates to CICS resources such as VSAM files. When COMMIT(CICS) is specified as the recovery option during configuration of Oracle Access Manager for CICS, Oracle Access Manager for CICS participates in SYNCPOINT processing. You must also

use CICS SYNCPOINT functions in your programs instead of Oracle9*i* COMMIT or ROLLBACK.

If there is more than one explicit connect (i.e., exec sql connect) within the same CICS 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 a connect statement is given control by a CICS LINK or XCTL within the same CICS transaction.

Passing Control with CICS LINK or XCTL Commands

Programs that use CICS LINK or XCTL commands must specify RELEASE_CURSOR=YES as an Oracle Precompiler option.

Explicitly Opened Cursors

Programs that explicitly open cursors must explicitly close them before control is passed to another program with CICS LINK, CICS XCTL, or as a program linked to a CICS RETURN.

- Each explicitly opened cursor is the target of a CLOSE statement.
- If COMMIT(ORACLE) is specified as the recovery option, then the Oracle COMMIT WORK RELEASE statement can be used.

Accessing Multiple Oracle9*i* Databases

Each program targeted to CICS can only communicate with one Oracle Access Manager for CICS and only one server because it can only be linked with one ORACSTUB interface stub. Access to more than one Oracle9*i* database server from a single CICS region is 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 do this, you first design your application so that the access logic for each database is contained in a separate CICS program. Then you configure an Oracle Access Manager and a corresponding ORACSTUB interface stub for each distinct server to be accessed. Finally, you link each of the CICS programs with the appropriate ORACSTUB stub.

Additional SQL Statement Restrictions

Programs used with Oracle Access Manager for CICS can only use Oracle data manipulation language (DML) SQL statements. Programs attempting to use data

definition language (DDL) or data control language (DCL) statements, such as CREATE TABLE or GRANT, receive an Oracle error code indicating improper use of distributed transaction mechanics.

IMS TM Programs

Oracle Access Manager for IMS TM allows COBOL or C programming language programs running in an IMS MPP, IFP, or BMP region to access an Oracle9*i* database server anywhere in your network.

From an IMS TM perspective, transaction programs accessing Oracle9*i* databases have no special requirements. The normal IMS calls for input and output message processing, data access, and synchronization behave as they would if Oracle9*i* were not in use.

From an Oracle9*i* perspective, Oracle Precompiler applications running under IMS TM are no different from other Oracle Precompiler applications. They contain SQL statements that are translated by the Oracle Precompilers and passed to the Oracle9*i* database server as the program runs. When using Pro*C or Pro*COBOL programs in an IMS TM transaction, however, there are some special considerations.

CONNECT Not Supported

A typical Oracle9*i* 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 in one of the following ways:

- The language interface token (LIT) linked with your application designates which Oracle Access Manager for IMS TM instance and which database is to be accessed. The name of this module is determined by your IMS TM system administrator.
- The Oracle user id and password, if any, are related to the IMS PSB name for the application.

Accessing Multiple Oracle9*i* 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 Oracle9*i* 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

Programs used with Oracle Access Manager for IMS TM can only use Oracle DML SQL statements. Programs attempting to use DDL or DCL statements, such as CREATE TABLE or GRANT, receive an Oracle error code indicating improper use of distributed mechanics.

A typical Oracle9*i* 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.

Accessing Oracle9*i* 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.

Controlling Oracle SQL Processing

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.

Processing of Oracle9*i* 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 Oracle9*i* 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.

Sample JCL

Oracle Precompilers are normally run in a batch environment but can be run under TSO if the required data sets have been allocated before the precompiler is run.

The following sample JCL illustrates the precompile, compile, and link steps necessary to build a precompiler application program.

> **Note:** Unlike previous releases of OS/390 Precompilers, this release relies on and exploits native support for long, mixed-case function names. The object produced will cause the binder to create a PM3 type program object. If this is not appropriate, other methods are available. See Appendix A, "API Short Name Support".

```
//PRECOMPL EXEC PGM=pcc_mod,               *- Refer to Note 1 -*
//             PARM='++/DD/SYSPARM'        *- Refer to Note 2 -*
//STEPLIB  DD  DSN=oran.orav.CMDLOAD,DISP=SHR
//ORA$LIB  DD  DSN=oran.orav.MESG,DISP=SHR
//SYSPRINT DD  SYSOUT=*
//SYSLIB   DD  DSN=oran.orav.H,DISP=SHR
//         DD  DSN=SYS1.SCEEH.H
//         DD  DSN=SYS1.SCEEH.SYS.H
//PROCFG   DD  DUMMY                     *- Refer to Note 12 -*
//SYSIN    DD  DSN=                      *- Refer to Note 4 -*,DISP=SHR
//SYSPUNCH DD  DSN=&&PCCOUT,DISP=(,PASS,DELETE),
//             UNIT=VIO                  *- Refer to Note 5 -*
```

```
//SYSPARM  DD  *
  INAME=/DD/SYSIN
  ONAME=/DD/SYSPUNCH
  LNAME=/DD/SYSPRINT
/*
//SYSOUT   DD  SYSOUT=*
//SYSERR   DD  SYSOUT=*
//SYSUT1   DD  UNIT=VIO                  *- Refer to Note 6 -*
//SYSUT2   DD  UNIT=VIO
//SYSUT3   DD  UNIT=VIO                  *- Refer to Note 3 -*
//*
If CICS is the target environment, run the CICS translator compiler here.  The
input to the CICS translator should be data set "&&PCCOUT" from the precompiler
step, and the output should be passed to the host language compiler (next).
//*
Run the Oracle Precompiler program's native language here.  The input to the
compiler should be data set "&&PCCOUT" from the precompile step (or the output
from the CICS translator step if CICS is the target environment).
//*
//LKED     EXEC PGM=IEWL,
//             PARM='XREF,LET,LIST,DYNAM=DLL'
//SYSPRINT DD  SYSOUT=*
//SYSLIB   DD  DSN=oran.orav.SQLLIB,DISP=SHR
//         DD  DSN=SYS1.SCEELKED
//         DD                            *- Refer to Note 7 -*
//SYSLMOD  DD                            *- Refer to Note 8 -*
//TEXT     DD                            *- Refer to Note 9 -*
//SYSUT1   DD  UNIT=VIO /
//SYSLIN   DD  *
  INCLUDE SYSLIB(stub)                   *- Refer to Note 10 -*
  INCLUDE TEXT
                                         *- Refer to Note 11 -*
/*
```

Usage Notes

The following usage notes apply to the previous JCL example:

1. This is the name of the Oracle Precompiler load module. There is a different Oracle Precompiler for each supported language. They are:

Language	Oracle Precompiler Load Module
C programming language	PROC

Language	Oracle Precompiler Load Module
COBOL	PROCOB18 (version 1 precompiler) PROCOB (version 9 precompiler)
PL/I	PROPLI

Note: PROCOB18 is an older precompiler release which only supports the functionality of the Oracle V7 database server. PROCOB is a newer precompiler release that supports all the latest Oracle9*i* functionality.

2. This PARM keyword shows an example of using parameter redirection. It points to the SYSPARM DD statement, which contains the INAME, ONAME, and LNAME parameters.
3. Only PROCOB and PROC require this DD statement.
4. This DD statement points to your Oracle Precompiler program source.
5. The output from the precompilation process is placed in this data set.
6. If the ORECLEN parameter is larger than 132, then the DCB attributes must be specified in the SYSUT1 or SYSUT2 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)
   ```

7. Include additional libraries required by your program and any runtime libraries required by the native language.
8. This DD statement describes the library in which your linkedited Oracle load module is placed.
9. This DD statement points to the object output from your Oracle Precompiler program's native language compiler.
10. The specific contents of the SYSLIN DD statement depend on the target environment. Refer to "Linking Your Program" on page 7-13 for more information.
11. Include any additional linkage editor statements you might require.
12. Only PROCOB requires this DD statement.

Precompiling Your Program

The following sections discuss precompiling your program using Oracle Precompilers.

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 OS/390 allows for parameters initially passed to user programs (for example, from JCL or under TSO). 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 the appropriate Oracle Precompiler manual for information on the EXEC ORACLE OPTION statement.
- Use a parameter redirection operator to direct the Oracle Precompiler to retrieve parameters from an OS/390 file in addition to the OS/390 parameter field. Refer to Chapter 3, "Oracle9i Utilities and OS/390 Files" for more information about parameter redirection.

When running under OS/390, some of the Oracle Precompiler options require special attention:

INAME specifies the input data set containing the source code, in any supported language, for the precompiler. Refer to Chapter 3, "Oracle9i Utilities and OS/390 Files" for a description of how to specify OS/390 data set names to the Oracle9*i* database server. 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 data set to contain the output from the Oracle Precompiler. Refer to Chapter 3, "Oracle9i Utilities and OS/390 Files" for rules on specifying OS/390 data set names to the Oracle9*i* database server. 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 data set to contain the listing from the Oracle Precompiler. Typically, this operand is coded as shown in the following example with a SYSPRINT DD statement pointing to a SYSOUT data set:

```
LNAME=/DD/SYSPRINT
```

CONFIG Pro*C and Pro*COBOL now support system configuration files, pointed to by PROCFG DD. Additionally, an overriding configuration file can be specified in SYPARM by /DD/CONFIG. The methods previously in place for specifying Oracle Precompilers option can also be used.

For example:

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

When specifying Oracle Precompiler options, separate each option with one or more blanks. Do not use a comma as a separator. If you use a data set to pass options to an Oracle Precompiler, then the data set must not have sequence numbers. If sequence numbers are found, then the Oracle Precompiler stops processing. To specify a fully-qualified data set name in an Oracle Precompiler option, use the /DSN/ or /DD/ specifications. Do not use a quoted data set name.

Return Codes

The Oracle Precompiler utility sets a return code of 0 upon successful precompilation of source code. A return code of 8 is set if the precompiler detects an irrecoverable error condition. In this case, an error message is written to the data set specified in the LNAME parameter passed to the Oracle 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

Refer to Chapter 2 of the *Oracle9i Enterprise Edition Installation Guide for OS/390* for specific compilers supported with each precompiler.

Pro*COBOL

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

- RETURN-CODE special register
- INTEGER values
- Compiler support
- MAXLITERAL option
- DYNAM compiler option

RETURN-CODE Special Register OS/390 COBOL programs use the RETURN-CODE special register to pass a return code from a main program back to OS/390. It is then used to form the job step completion code. The OS/390 standard subroutine linkage places unpredictable values into the RETURN-CODE special register as a subroutine is called. OS/390 subprograms 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 OS/390, then the unpredictable value remains in the RETURN-CODE special register and is used to form the job step completion code. You are responsible for ensuring the correctness of the RETURN-CODE special register.

INTEGER Values In the generic Oracle documentation, SQL library function parameters are often documented with a data type of INTEGER. In OS/390, integers are a fullword and must be defined in COBOL as S9(8) COMP fields. An S9(4) 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 OS/390. 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.

MAXLiteral Oracle Precompiler Option The default value of the MAXLITERAL Oracle Precompiler option is 120. If you supply a MAXLITERAL value, then it is rounded down to the next full word boundary (for example, 50 to 48).

DYNAM Compiler Option The Oracle Precompiler interface stub must be statically linked with your application. Therefore, the COBOL compiler DYNAM option must not be used with Pro*COBOL programs. You can dynamically call your own subroutines by using the CALL `identifier` form of the COBOL CALL statement. Refer to the appropriate compiler documentation for more information.

Pro*C

When using Pro*C, each of the following have special considerations:

- Object support

Object Support The Object Type Translator (OTT) utility is used to convert database object type definitions into C programming language declarations that are included in your Pro*C applications.

Pro*PL/1

The native language output from the Pro*PL/1 Precompiler that is directed to the data set specified in the ONAME parameter cannot have a RECFM of V. This is because an OS/390 compiler restriction requires all PL/I programs contained on a variable length data set have a blank or a sequence number in columns 1 to 8. This restriction does not apply to the data set input to PROPLI, only to the intermediate data set passed to the PL/I compiler.

Compiling Your Program

After your program has been precompiled, the resulting source file is passed to the host language compiler.

CICS Programs

You must run the Oracle Precompiler program first and pass the output to the CICS translator to translate the CICS commands. Otherwise, the CICS translator issues warning messages for each EXEC SQL statement that is encountered. Refer to the machine-readable samples in the SRCLIB library for the JCL used to process COBOL or C programming language programs.

Linking Your Program

After your program has been compiled, the resulting object code is passed to the linkage editor where it is linked with the Oracle Precompiler interface stub. Each

target environment has a unique stub and different linking considerations apply to each of the target environments. In all cases the interface stub must be linked directly to your load module. Dynamic loading techniques (for example, the COBOL dynamic call mechanism) cannot be used with the interface stub.

Batch and TSO Programs

When linking a 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 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 stub routine. You can contact Oracle Support Services for additional assistance.

CICS Programs

When linking a program to run under CICS, add the following linkage editor control statement to the SYSLIN DD statement:

```
INCLUDE SQLLIB(ORACSTUB)
```

Do not include references for ORASTBL or AMILS.

AMODE must be set to 31. There are no special requirements for RMODE.

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

IMS TM Programs

When 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 module described later in this chapter. The IMS TM system administrator generates and names the `lit-module` according to installation standards.

Do not include references to ORASTBL or ORACSTUB.

If there are unresolved external references for symbols whose names begin with SQL, then ensure AMILS is included in the linkedit. This 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 Your Program

After your program has been linked, the resulting load module is run in the target environment. As your program runs, it passes SQL statements to the SQL interface stub that is linked with it. The interface stub contains logic to dynamically locate the full Oracle SQL interface for the target environment and pass control to it. The full interface must be available in a load library as your program runs. Depending on the target environment, other runtime considerations might apply. These are discussed in the following sections:

- UNIX System Services (USS)
- Batch and TSO Programs
- CICS Programs
- IMS TM Programs

UNIX System Services

Pro*C, PRO*COB, and PROCOB18 are supported in the UNIX System Services environment. Refer to Chapter 4, "Accessing Oracle9i Under USS", for more information.

Administering Pro*COB Under UNIX System Services

Pro*COB parses the input files, and must know the include pathname to find all the `include` files. The Pro*COB command line must have each nonstandard include pathname specified. For example:

```
$procob iname=xyz.pco include=/home/jones/copybooks oname=xyz.cbl
$/usr/lpp/cobol/bin/cob2 -c xyz.cbl
```

Building Pro*COB Programs You can build your own Pro*COB programs with `demo_procob.mk`. For example, to precompile, compile, and link the userprog.pco program, enter:

```
$ make -f $ORACLE_HOME/precomp/demo/proc/demo_proc.mk userprog
```

Users must change makefiles or scripts that call `proc`. The `proc` command line must have each nonstandard include pathname specified. For example:

Sample Programs Please note that the sample programs require that you run a SQL script to create the various tables and packages required by the precompiler. You must run the SQL scripts in the precomp/demo/sql directory for the following sample programs in precomp/demo/:

```
lobdemo1.pco
```

To build one of the sample PRO*COB programs, cd to the precomp/demo/proc directory and issue the following make command:

```
make -f demo_proc.mk sample1
```

To build all of the sample PRO*COB programs, cd to the precomp/demo/proc directory and issue the following make command:

```
make -f demo_proc.mk samples
```

Note: If your program depends on non-Oracle libraries, you may have to alter `demo_procob.mk` to include them.

Administering Pro*C Under UNIX System Services

Pro*C is upwardly compatible for all Pro*C programs written in C. (Pro*C does not support C++ programs.) Pro*C parses the input files, and must know the `include` pathname to find all the `include` files.

Users must change makefiles or scripts that call `proc`. The `proc` command line must have each nonstandard include pathname specified. For example:

```
$proc iname=xyz.pc include=/home/jones/include
$c89 -r/home/jones/include xyz.c
```

Building Pro*C Programs

You can build your own Pro*C programs with `demo_proc.mk`. For example, to precompile, compile, and link the `userprog.pc` program, enter:

```
$ make -f $ORACLE_HOME/precomp/demo/proc/demo_proc.mk userprog
```

> **Note:** If your program depends on non-Oracle libraries, you may have to alter `demo_proc.mk` to include them.

Sample Programs

Please note that the sample programs require that you run a SQL script to create the various tables and packages required by the precompiler. You must run the SQL scripts in the `precomp/demo/sql` directory for the following sample programs in `precomp/demo/`:

```
sample11
cv_demo
ansidyn1

coldemo1
lobdemo1
objdemo1
navdemo1

cppdemo2
```

To build one of the sample PRO*C programs, cd to the `precomp/demo/proc` directory and issue the following make command:

```
make -f demo_proc.mk sample1
```

To build all of the sample PRO*C programs, cd to the `precomp/demo/proc` directory and issue the following make command:

```
make -f demo_proc.mk samples
```

To build one of the C++ sample PRO*C programs, cd to the `precomp/demo/proc` directory and issue the following make command:

```
make -f demo_proc.mk cppdemo1
```

To build all of the C++ sample PRO*C programs, cd to the `precomp/demo/proc` directory and issue the following make command:

```
make -f demo_proc.mk cppsamples
```

To build one of the Object sample PRO*C programs, cd to the `precomp/demo/proc` directory and issue the following make command:

```
make -f demo_proc.mk coldemo1
```

To build all of the Object sample PRO*C programs, cd to the `precomp/demo/proc` directory and issue the following make command:

```
make -f demo_proc.mk object_samples
```

Batch and TSO Programs

As your batch or TSO program runs, it must have access to several load modules contained in the *oran.orav*.CMDLOAD library and all modules in the *oran.orav*.MESG library. This includes the following modules:

LIBCLNTS — is the SQL and OCI interface module. This is a DLL containing the SQL and OCI interfaces. This module also serves as the "shared library" used by Oracle tools and utilities.

All Modules in MESG data set.

Your programs can access these modules in the following ways:

- Add a JOBLIB or STEPLIB statement to your JCL and point to the *oran.orav*.CMDLOAD library.
- Add an ORA$LIB statement to your JCL and point to the library, *oran.orav*.MESG. Message modules will be accessed from here.
- Your Oracle system administrator might place an *oran.orav*.CMDLOAD library in the system LINKLIST or he might place various modules in the PLPA or EPLPA. If this is the case, then you might have other directions for writing your JCL.

Note: One common option you can use is to put CMDLOAD data set in STEPLIB and MESG data set in ORA$LIB.

If your program receives OS/390 message IEA703I, then this is an indication that one of the required modules cannot be located. IEA703I is accompanied by error codes 106-n, 306-n, or 806-n. Refer to the IBM *OS/390 System Codes* manual to determine why LIBCLNTS cannot be located and correct the problem.

CICS Programs

Normal CICS practices for defining your program to CICS are followed for Oracle Precompiler programs. CICS programs access a shared copy of the full SQL interface (called the CICS 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. You need to contact your CICS system administrator to have this condition corrected.

IMS TM Programs

Normal IMS TM practices for defining your program to IMS TM 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. You must contact your IMS TM system administrator to have this condition corrected.

8

Oracle Call Interface

Oracle9*i* for OS/390 supports two types of programmatic interfaces: Oracle Precompilers and the Oracle Call Interface (OCI). Together, they allow precise control over SQL statement operations in a high-level source program.

Read this chapter for OS/390-specific information about using the Oracle Call Interface under OS/390 and Chapter 7, "Oracle Precompilers" for information about Oracle Precompilers. Sections in this chapter include:

The product-specific documentation for OCI is contained in the *OCI Programmer's Guide.*

Oracle Call Interface Overview

The Oracle Call Interface (OCI) provides precise control over all aspects of Oracle9*i* database access to your C programming language application programs. Unlike Oracle Precompiler applications that use Pro*C to convert embedded SQL statements into C programming language statements and function calls, OCI applications are written to specify the C programming language function calls directly. Oracle9*i* OCI only supports programs written in C programming language. Although use of Oracle V7 OCI is still supported for existing applications and existing applications can be run without relinking, it is no longer documented.

OS/390 supports several major operating environments and OCI programs can be targeted to run in any of them. Whether your application requires traditional batch processing, interactive processing under TSO, or transaction processing under CICS or IMS TM, OCI can be a part of your application architecture.

OCI is also supported in the OS/390 UNIX System Services environment. Refer to Chapter 4, "Accessing Oracle9i Under USS", for additional information about building and running OCI programs in the OS/390 UNIX System Services environment.

Regardless of which environment you choose, the same steps are done. These steps are:

1. Compiling the program with the language's compiler.
2. Linking the program with an Oracle interface stub designed especially for the target environment. Batch, TSO, CICS, and IMS TM each require linking with a different stub.
3. Running your program in the target environment.

This chapter discusses each of the steps needed to build and run an application program that uses Oracle9*i* OCI. It contains environment-specific program design considerations. For more information, refer to the *OCI Programmer's Guide*.

Target Environment Design Considerations

This section discusses application design considerations that apply specifically to a target environment.

TSO Programs

To use an OCI application under TSO (by the CALL command or as a command processor), ensure you call OCISessionEnd or OCIServerDetach before the program ends. Otherwise, an abend with completion code A03 occurs when the program attempts to end. The OCISessionEnd or OCIServerDetach call ensures Oracle cleanup actions are taken, in particular the removal of an OS/390 subtask that is used to handle TSO session attention signals.

The IBM TSO TEST processor does not support cross memory mode operations. Because Oracle applications perform all database operations in cross memory mode, you cannot escape to TSO TEST when an Oracle application is performing operations within the database.

CICS Programs

Oracle OCI calls are not supported by Oracle Access Manager for CICS.

IMS TM Programs

Oracle Access Manager for IMS TM allows OCI programs written in C programming language and running in an IMS MPP, IFP, or BMP region to access an Oracle9*i* database server anywhere in your network. Programs can access and update IMS, DB2, and Oracle9*i* data within a single synchronized transaction.

From an IMS TM perspective, transaction programs accessing Oracle9*i* databases have no special requirements. The normal IMS calls for input and output messages processing, data access, and synchronization perform as if the Oracle9*i* system is not being used.

From an Oracle9*i* perspective, OCI applications running under IMS TM are no different from other OCI applications. They contain OCI function calls that are passed to the Oracle9*i* database server as the program runs. When using OCI programs in an IMS TM transaction, there are some special considerations.

Accessing Multiple Oracle9*i* Databases

Each program target to IMS TM can only communicate with one Oracle Access Manager for IMS TM and one server because it can only link with one LIT. Access to more than one Oracle9*i* database server from a single IMS TM region can be done 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 do this, you first design your application so that the access logic for each database is contained in a separate IMS TM program. You then 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 Restricted OCI Functions

Programs used with Oracle Access Manager for IMS TM can only use Oracle DML SQL statements. Programs attempting to use DDL or DCL statements, such as CREATE TABLE or GRANT, receive an Oracle error code indicating incorrect use of distributed transaction mechanics.

A typical Oracle9*i* application might use OCI functions for commit or rollback statements to control whether database updates are committed or removed. With Oracle Access Manager for IMS TM, these OCI functions 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.

Unavailable Calls

When using Oracle Access Manager for IMS TM, the following OCI function calls are not available:

- All OCI security calls
- All OCI thread calls
- All OCI direct path load operations
- The following function calls:

- OCIBreak
- OCIDefineDynamic
- OCIEnvCreate
- OCIEnvInit
- OCIInitialize
- OCILdaToSvcCtx
- OCILogoff
- OCILogon
- OCIPasswordChange
- OCIReset
- OCIServerAttach
- OCIServerDetach
- OCISessionBegin
- OCISessionEnd
- OCISvcCtxToLda
- OCITerminate
- OCITransCommit
- OCITransDetach
- OCITransForget
- OCITransPrepare
- OCITransRollback
- OCITransStart

Oracle Access Manager for IMS TM supports both version 7 and version 9 OCI functions. OCON and OCOF are only supported for version 7. For version 9, use OCIStmtExecute with mode parameter OCI_COMMIT_ON_SUCCESS. If you specify OCI_COMMIT_ON_SUCCESS with version 7, then the mode parameter is changed to OCI_DEFAULT.

Accessing Oracle9*i* 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 Oracle9*i* 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 Oracle9*i* 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. The application developer and IMS administrator decide which REO to use. The REO is not specified by or in the application program. It is an Oracle Access Manager for IMS TM configuration parameter.

Compiling Your Program

When compiling your program there are considerations for long name support and callback restrictions.

Long Name Support

Unlike previous releases of the OS/390 implementation of OCI, this release relies on and exploits native support for long, mixed-case function names. So, when compiling an OCI application, the LONGNAME compiler option must be specified. The object produced will cause the binder to create a PM3 type program object. If this is not appropriate, another method is available. See Appendix A, "API Short Name Support".

Callback Restriction

Applications using OCI functions that make use of callbacks must be compiled with the DLL option.

UNIX System Services

The Oracle Call Interface may be used in C programs developed for the USS environment. Refer to Chapter 4, "Accessing Oracle9i Under USS" for more information.

To build one of the sample OCI programs, cd to the `rdbms/demo` directory and issue the following make command:

```
make -f demo_rdbms.mk build EXE=demo OBJS="demo.o ..."
```

For example:

```
make -f demo_rdbms.mk build EXE=oci02 OBJS=oci02.o
```

To build all of the sample OCI programs, cd to the `rdbms/demo` directory and issue the following make command:

```
make -f demo_rdbms.mk demos
```

Note: The sample OCI C++ programs are not supported in the USS environment due to a current limitation with the IBM C++ compiler on OS/390. The current versions of the C++ compiler (version 2.6 through version 2.9) do not support the "long long" datatype.

Linking Your Program

After your program has been compiled, the resulting object code file is passed to the linkage editor where it is linked with the Oracle interface stub. Each target environment has a unique stub and different linking considerations apply to each target environment. In all cases the interface stub must be linked directly to your load module. Dynamic loading cannot be used with the interface stub.

Sample Link JCL

Following is an example of JCL that might be used for linking:

```
//LKED     EXEC PGM=IEWL,
```

```
//         PARM='XREF,LET,LIST,DYNAM=DLL'
//SYSPRINT DD  SYSOUT=*
//SYSLIB   DD  DSN=SYS1.SCEELKED,DISP=SHR
//         DD  *- Refer to Note 1 -*
//         DD  SYS.SCEELKED
//SQLLIB   DD  DSN=oran.orav.SQLLIB,DISP=SHR
//SYSLMOD  DD  *- Refer to Note 2 -*
//TEXT DD  *- Refer to Note 3 -*
//SYSUT1   DD  UNIT=VIO
//SYSLIN   DD  *
INCLUDE SQLLIB(stub) *- Refer to Note 4 -*
INCLUDE TEXT
*- Refer to Note 5 -*
/*
```

Usage Notes

The following usage notes apply to the previous JCL example:

1. Include additional libraries required by your program and any runtime libraries required by the C programming language compiler.
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 the C programming language compiler's program native language compiler.
4. The specific contents of the SYSLIN DD statement depend on the target environment. These contents are discussed later in this chapter.
5. Include any additional linkage editor statements you require.

Batch and TSO Programs

When linking an OCI program to run in batch or TSO, you must add the following linkage editor control statement to the SYSLIN DD statement:

```
INCLUDE SYSLIB(ORASTBL)
```

The ORASTBL stub provides long name support for OCI functions and support for SQL functions.

If there are external references for symbols whose names begin with the letter O, then the linkage editor cannot find one or more of the function calls in your application. Ensure ORASTBL is included in your linkedit. The ORASTBL stub

resolves any calls in your code. If ORASTBL 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.

IMS TM Programs

When linking a program to run under IMS TM, add the following linkage editor control statements to the SYSLIN DD statement:

```
INCLUDE SQLLIB(AMILS)
INCLUDE SQLLIB(lit-module)
```

The AMILS module must be the version that is compatible with this release if Oracle9*i* OCI is used.

The `lit-module` statement refers to a customer-generated language interface module. The IMS TM system administrator generates and names the statement according to the installation's standards.

Do not include references to ORASTBL, ORASTUBS, or ORACSTUB.

If there are external references for symbols whose names begin with the letter O, then the linkage editor cannot find one or more of the function calls in your application. Ensure AMILS is included in your linkedit. The AMILS stub resolves any calls in your 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 Your Program

After your program has been linked, the resulting load module is run in the target environment. As your program runs, it passes OCI function calls to the OCI interface stub linked with it. The interface stub contains logic to dynamically locate the full Oracle OCI interface for the target environment and to pass control to it. The full interface must be available in a load library as your program runs. Depending on the target environment, other runtime considerations might apply.

Batch and TSO Programs

The requirement for OCI programs in batch and TSO environments is the same as Precompiler programs. Refer to the "Running Your Program" on page 7-15.

IMS TM Programs

Normal IMS TM practices for defining your program to IMS TM are followed for OCI programs. IMS TM programs access a shared copy of the full OCI interface that must be configured and available before your program begins running. If the interface is not available, then you receive error message 3042 from IMS TM. You must contact your IMS TM system administrator to have this condition corrected.

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 for OS/390 server 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 OS/390 clients running under USS as well as remote clients on any multi-threaded platform. It excludes native TSO/batch and IMS clients at this time

The TCP connectivity requirement does not mean that the client must be connected to the server via Oracle Net PROTOCOL=TCP. The client can be connected via any protocol (including Oracle Net PROTOCOL=XM for a local USS 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.

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

9

SQL*Plus

The primary interface to the Oracle9*i* database server, SQL*Plus, provides a powerful environment for querying, defining, and controlling data.

This chapter describes accessing and running SQL*Plus commands and procedures in an OS/390 environment. The following topics are discussed:

The information in this chapter supplements the primary documentation for SQL*Plus found in the *SQL*Plus User's Guide and Reference*.

Running Under UNIX System Services

Oracle9*i* provides a native OS/390 UNIX System Services version of SQL*Plus. To run it, first ensure the environment is established as described in Chapter 4, Running an Oracle Utility Under USS, then enter the following at the command prompt:

```
sqlplus
```

For additional information, refer to Chapter 4, "Accessing Oracle9i Under USS" and the *SQL*Plus User's Guide and Reference*.

Running Under TSO

To access SQL*Plus as a command processor, use the following syntax:

```
SQLPLUS [-S] [userid[/password[@connect-string]]] [@start_file]
```

where:

-S	calls SQL*Plus silently. The initial banner and the command prompts are not displayed. If used, then -S must be specified immediately following the SQLPLUS command.
`userid`	is the Oracle user id.
`password`	is the Oracle password associated with `userid`.
`connect-string`	is a tnsnames alias name or a complete Oracle Net address string. This parameter is optional if the Oracle database server is specified using methods described in Chapter 2. Otherwise, refer to Chapter 10 for further details.
`start_file`	specifies a SQL command file to start after SQL*Plus initialization. This string must be separated from `userid` and `password` by at least one blank to indicate it is a start file and not a database pointer.

All parameters are optional to SQL*Plus.

According to the following syntax, the command connects Oracle user SCOTT to the Oracle instance specified in the ORA1 tnsnames alias. This automatically runs the SQL statements contained in member INITIAL in data set TEST.ORACLE.SQL:

```
SQLPLUS SCOTT/TIGER@ORA1 @/DSN/TEST.ORACLE.SQL(INITIAL)
```

Attention Processing

Use of the [PA1] or [Attn] key interrupts the operation of SQL*Plus. When you enter a command producing results you did not expect or are not interested in viewing, using the attention interrupt is useful.

When running in TSO/E, SQL*Plus attempts to establish an attention processing exit using the TSO/E STAX service. For this to work, the ORASTAX module must be loadable from JOBLIB/STEPLIB, a linklist library, or LPA. If ORASTAX cannot be loaded, attention processing is suppressed; no messages are issued to indicate this condition.

When attention processing is established, it is used by both SQL*Plus and by the Oracle program interface code. The effect of an attention signal depends on whether SQL*Plus or an Oracle server is in control. If a request has been issued to a server, the attention signal is associated with the Oracle program interface and it will halt the in-progress server request. Otherwise, the signal is handled by SQL*Plus itself and will halt the current SQL*Plus activity. For example, if SQL*Plus is displaying large amounts of data from a SELECT statement, and attention will halt the output display and return to the "SQL>" prompt.

Processing States

Because the computer operator can issue an attention interrupt at any time, the attention exit responds differently depending upon the processing state of SQL*Plus. Regardless of the processing state, the following message is displayed:

```
!
Oracle ATTN.
```

If the [PA1] key is pressed during a running query or update, then the operation is canceled. The Oracle9*i* database server returns the following error message, acknowledging the cancel:

```
Error: ORA-1013: user requested cancel of current operation
```

If the [Attn] key is pressed during a running query, then the query might continue to retrieve and display a number of rows before the cancel is acknowledged. The query sometimes continues because the [Attn] key operates asynchronously. Any messages queued to the user before the attention interrupt are displayed.

If there is no update or query running, then SQL*Plus waits for other user input.

Batch Processing

Because attention processing cannot occur in the OS/390 batch environment, attention processing considerations do not apply to SQL*Plus operations in batch.

Running in Batch

You can call SQL*Plus in the batch environment by running the ORASQL JCL procedure. This procedure is created during the Oracle9*i* database server installation process. Consult with your DBA or systems programming staff for the location of this procedure. A copy of the procedure is reproduced in the following example:

```
//ORASQL   PROC  INDEX=oran,
//               LIBV=orav,
//               SYSOUT='SYSOUT=*'
//ORASQL   EXEC  PGM=SQLPLUS,REGION=4M
//STEPLIB  DD    DSN=&INDEX..&LIBV..CMDLOAD,DISP=SHR
//ORA$LIB  DD    DSN=&INDEX..&LIBV..MESG,DISP=SHR
```

When running this JCL procedure, include additional DD statements referring to other data sets required for proper operation of the utility.

The following JCL illustrates the use of the procedure described in the previous example:

```
//USER1JOB JOB   ........
//STEP1    EXEC  ORASQL,PARM='SCOTT/TIGER'
//ORA@sid  DD    DUMMY
//SYSIN    DD    DSN=SCOTT.ORACLE.SQL(SQLSTUFF),DISP=SHR
```

Note the additional DD statements:

- `ORA@sid`, where `sid` is the SID associated with the Oracle9*i* database server to which you want to connect.
- SYSIN, which contains the SQL commands you want to run.
- SYSOUT and SYSERR will be used if present.

Use this procedure as a model for creating your own JCL procedures conforming to your installation's policies.

To pass parameters to SQL*Plus in the batch environment, use the JCL PARM field.

You can also access the Oracle9*i* database server through the terminal monitor program (TMP) in batch mode. Refer to "Running Under TSO" on page 9-2 in this case. File allocations required in TSO access are also required in batch mode.

SQL*Plus Profiles

During initialization, SQL*Plus searches for an allocation to the SQLLOGIN DD statement. If a file is allocated to this DD statement, then it is opened for input and the contents are run automatically, one command at a time, until an EOF is encountered. This DD statement points to a file that can contain any valid SQL or SQL*Plus statements you want run with each logon to SQL*Plus.

If the file allocated by the SQLLOGIN DD statement is not found during initialization, then the following messages are displayed:

```
ERROR OPENING /DD/SQLLOGIN: DDNAME STATEMENT MISSING OR MISSPELLED
UNABLE TO OPEN FILE: "/DD/SQLLOGIN"
```

These messages signal an unsuccessful SQLLOGIN allocation and you are provided with the normal SQL prompt. This message display is not an error condition. It indicates automatic running of SQL or SQL*Plus statements at startup did not occur.

If you are not using automatic running of SQL or SQL*Plus statements at startup and do not want to receive these messages, then use the DUMMY parameter in the data set allocation.

A short SQLLOGIN profile example is shown in the following example:

```
Set echo off
set feedback 4
set message on
set echo on
select * from daily_reminder;
```

SQL*Plus HOST Command

The SQL*Plus HOST command allows you to use a variety of commands and utilities from within SQL*Plus.

Running TSO/E Commands

The SQL*Plus HOST command lets you run TSO/E commands from within SQL*Plus:

```
SQL> HOST SEND 'GOOD MORNING' USER(SMITH)
```

The HOST command is only valid in native TSO/E and cannot be used in batch mode.

Calling CLISTs

The SQL*Plus HOST command lets you use CLISTs from within SQL*Plus. To run a CLIST use the following, where `clist` is the name of the CLIST to run:

```
SQL> HOST clist
```

Calling OS/390 Editors

Call an OS/390 editor using one of the following SQL*Plus HOST commands:

```
SQL> HOST EDIT filename OLD DATA
SQL> HOST ISPF 2
```

Multiple SQL*Plus Processes

Use the HOST command from within SQL*Plus with a different Oracle user id to start a new, separate SQL*Plus process. Other Oracle utilities can also be called, subject to available memory.

SQL*Plus Time Usage Information

SQL*Plus provides two means of gathering time usage information: the TIMING and SET TIMING commands. Refer to the *SQL*Plus User's Guide and Reference* for more information about TIMING and SET TIMING.

TIMING Command

The TIMING command is used to record and display time usage statistics. The timing data returned has the following format:

```
Elapsed x:xx:xx.xx,  user-CPU    y.yy sec
```

where:

`x:xx:xx.xx`	is the amount of time since the current timing area started.
`y.yy`	is the amount of CPU-time used since the current timing area started.

SET TIMING Command

The SET TIMING command is used to display time usage information. The information returned by the SET TIMING command is displayed after each SQL*Plus command is run. The timing data returned has the following format:

```
Elapsed x:xx:xx.xx,  user-CPU    y.yy sec
```

where:

`x:xx:xx.xx`	is the amount of time since the SQL*Plus command was issued.
`y.yy`	is the amount of CPU-time used to run the SQL*Plus command.

Using OS/390 Editors from SQL*Plus

You can use the ISPF editor with SQL*Plus:

ISPF Editor

To support the ISPF editor, the editor must be defined as ISPF using a DEFINE_ EDITOR command from the command line or from a profile. Once the editor is defined as ISPF, when a SQL*Plus EDIT command is issued, the ISPLINK interface is dynamically loaded.

Data Set Enqueuing

When you begin editing a member with ISPF, ISPF obtains an exclusive ENQ for QNAME SPFEDIT and a 52-byte RNAME consisting of the data set name and member. ENQ is held during the entire edit session. When you save the member, ISPF obtains a second exclusive ENQ for QNAME SPFEDIT and a 44-byte RNAME containing only the data set name. ISPF opens the PDS for output, writes the edited member, and closes the PDS. The two enqueues are released in the order they were obtained.

Restricting User's Privileges in SQL*Plus

The DBA can use the PRODUCT_USER_PROFILE table, located in the SYSTEM account, to disable certain SQL*Plus commands for a user. You might be able to use the PRODUCT_USER_PROFILE table with your applications. For more information about the PRODUCT_USER_PROFILE table, consult with the DBA for your system and refer to the *SQL*Plus User's Guide and Reference.*

Exiting SQL*Plus

To exit SQL*Plus, you can enter one of the following:

```
SQL>EXIT [return code keyword | variable]
SQL>QUIT [return code keyword | variable]
```

QUIT is a synonym for the EXIT command.

The EXIT command lets you specify an operating system return code. This enables you to run SQL*Plus command files in batch mode and detect any unexpected events. Under OS/390, the return code values are:

Command	Condition	Return Code
SQL>EXIT	SUCCESS	0
	FAILURE	1
	WARNING	1
	`<variable>`	Value of `<variable>`
	SQL.SQLCODE	Return code from the last SQL operation to the database

The syntax for the EXIT command is described in the *SQL*Plus User's Guide and Reference.*

Spooling SQL*Plus Output

When using the SQL*Plus SPOOL command, SQL*Plus uses a default output file suffix of LST.

Usage Notes

There are special considerations for the following:

- Special characters
- String concatenation
- Input line truncation
- SQL*Plus ASCII function

Special Characters

The SQL symbol for negation is the exclamation point. For not equal, <> is recommended.

String Concatenation

The SQL symbol for concatenation under OS/390 is the solid vertical bar.

A string concatenation function, CONCAT, provides an alternative to using the vertical bar character for concatenation. Installations that do not have computer keyboards with the vertical bar can use this function. For installations where the vertical bar is available on the computer keyboards and where portability to environments with other character sets is not important, the vertical bar character can be used.

The CONCAT function has the following syntax, where `x` and `y` are the strings to concatenate:

```
CONCAT('x','y')
```

The quotes in the syntax are required.

For example, the following statement selects rows where the employee name is CLARK:

```
SELECT * FROM EMP WHERE ENAME = CONCAT('CL','ARK');
```

Input Line Truncation

SQL*Plus currently truncates, without warning, input lines exceeding 255 characters.

SQL ASCII Function

Under OS/390, the SQL ASCII function can cause confusion. Contrary to what the name implies, the ASCII function does not convert an EBCDIC value to its ASCII equivalent.

The ASCII function takes a character, in this case EBCDIC, and returns the number representation for that character in the given character set. For example, the ASCII value of A is 193 (ASCII('A')=193). The inverse function is CHR (meaning, CHR(193)='A').

Unsupported Functions

The following functions are not supported:

- SPOOL OUT
- SET NEWPAGE 0
- RUNFORM

SPOOL OUT

The SPOOL OUT function documented in the *SQL*Plus User's Guide and Reference* is not supported in OS/390 in the TSO or batch environments. To spool to a data set that is automatically printed, you can use the following steps:

1. Allocate a DDname to SYSOUT:

   ```
   //SPOOLED  DD SYSOUT=*
   ```

2. Spool the output of the query to the pre-allocated DDname:

   ```
   SQL> SPOOL /DD/SPOOLED
   ```

SET NEWPAGE 0

Using SET NEWPAGE 0 in SQL*Plus in OS/390 does not clear the screen between logical pages on a 3270 computer. NEWPAGE 0 only applies to printed output.

RUNFORM

The RUNFORM command is unavailable. When you enter the RUNFORM command, you receive an error message.

10

Oracle Net

Oracle Net is a component of the Oracle server providing distributed database and processing capabilities. Oracle Net for OS/390 supports network communications between Oracle applications and Oracle9*i* database server systems across different OS/390 systems or foreign operating systems.

This chapter discusses usage considerations for Oracle Net on OS/390 and covers OS/390 clients (including servers that initiate database links) accessing remote Oracle servers as well as remote clients accessing a Oracle server on OS/390. It contains the following sections:

For product-specific information, refer to the *Oracle9i Net Services Administrator's Guide*.

See Chapter 2, "Using the OS/390 Database Instance", for information about accessing an Oracle instance from a local client. For information about accessing a remote Oracle instance from USS, see Chapter 4, "Accessing Oracle9i Under USS".

Oracle Net Overview

The Oracle Net Network Service acts as the bridge between Oracle clients and servers and the communication facilities of the OS/390 operating system, thus supporting network communications between Oracle applications and Oracle9*i* database server systems or gateways across different images or operating systems. One protocol is available: TCP/IP. The Oracle Net Network Service is controlled via the parameters in its service definition.

Distributed Processing

Dividing processing between a front-end computer running an application and a back-end computer used by the application is known as distributed processing. Oracle Net enables an Oracle tool or application to connect to a remote computer supporting an Oracle9*i* database server or gateway.

Distributed Database

Several databases linked through a network, appearing to a user as a single logical database, are known as a distributed database. Oracle tools running on a client computer can share and obtain information retrieved from other remote Oracle9*i* database servers or gateway systems. Regardless of the number of database information sources, the user may only be aware of one logical database.

Oracle Net Terminology

The following terms are used to explain the architecture of Oracle Net for OS/390:

Host	is the computer on which the database resides. It runs the Oracle9*i* database server or an Oracle gateway.
Server	is the Oracle9*i* database service.
Client	is the application using Oracle Net to communicate with a server. A server is also considered a client if it initiates a connection with another server.
Protocol	is a set of standards governing the operation of a communication link.
Network	is a configuration of devices and software connected for information interchange.

Remote Access to OS/390 Server Using Oracle Net

Remote (inbound) clients access Oracle9*i* for OS/390 database instances through Oracle Net using Oracle Net address strings as follows:

1. Oracle Net listens on a single endpoint (network address) for each protocol. All remote clients that go through a particular network service with a particular protocol use the same network address regardless of which database instance they want to access. All TCP/IP clients specify the same hostname (or IP) and port number.

2. Clients indicate the target database instance that they want with the '(CONNECT_DATA=(SID=*sid*))' clause in the Oracle Net address string. This is required with Oracle Net for OS/390.

Oracle Net for OS/390 Filenames

Oracle Net for OS/390 supports a number of Oracle Net files that are used to specify TNSNAMES connect descriptors as well as Oracle Net parameters. The Oracle Net parameters are used to configure Oracle Net processing options for Oracle Net facilities such as Name Server, LDAP as well as Oracle Net logging.

The base Oracle Net documentation, *Oracle9i Net Services Administrator's Guide*, refers to files in the following form:

basename.extension

where basename is the product name and *extension* is the extension.

An example of this form is SQLNET.ORA.

These files are mapped to DDnames on OS/390. The following DDnames are implemented under OS/390:

SQLNET	defines a data set containing any SQLNET.ORA diagnostic, ASO, or Oracle Names parameters. It is not necessary to allocate this DD unless these features are desired. Refer to *Oracle9i Net Services Administrator's Guide* or the *Oracle Label Security Administrator's Guide* for more information.
SQLNETLG	defines a data set into which any logging output is written. Oracle Corporation recommends that this be defined as a SYSOUT data set in a held output class.
SQLNETTC	defines a data set into which any logging output is written. Oracle Corporation recommends that this be defined as a SYSOUT data set in a held output class.
TNSNAMES	defines a data set containing all the TNS connect descriptors and aliases for your installation. For further information on TNS connect descriptors, refer to the *Oracle9i Net Services Administrator's Guide*. This DDname is not necessary on server JCL unless DBLINKS originate from the server.
LDAP	defines the location of the LDAP server.
TNSNAV	TNS client navigation. (Generally not used on OS/390.)
INTCHG	Interchange. (Generally not used on OS/390.)

Locating the Oracle Net Service

Beginning with Release 9.2, clients open their own sockets and the SSN parameter used by clients in previous releases is no longer required.

Oracle Net Connect Descriptors for OS/390

The Oracle Net connect descriptors for OS/390 adopts the same syntax as the base Oracle Net connect descriptors.

TCP/IP Addresses

The syntax for the TCP/IP address portion of a connect descriptor on OS/390 is described as follows:

```
aliasname=
          (DESCRIPTION=
           (ADDRESS=
            (PROTOCOL = TCP)
            (HOST = hostname)
            (PORT = port_num)
            (CONNECT_DATA=(SID=sid)))
```

where:

aliasname	is the name used to reference this connection description.
hostname	is the host nickname as defined to TCP/IP or the host internet address as a decimal value.
port_num	is the TCP/IP port number on which the target database is listening.
sid	is the SID of the target Oracle server.

Examples

The examples are:

Connecting to a Remote Server Using Oracle Net

With Oracle9*i*, Release 2, clients open their own sockets and no longer need to direct their request through the Net address space. To use Oracle Net for OS/390 in client mode, the connect string must include an alias defined in a TNSNAMES file, LDAP, an Oracle Names server, or a valid TNS connect descriptor.

- To use the descriptors defined in the TNSNAMES file in a TSO session, allocate:

  ```
  ALLOC F(TNSNAMES) DA('ORA.SQLNET.CNTL(tnsname)') SHR REUSE
  ```

- To connect to a remote database with an alias of ORAUNIX using SQL*Plus, the following connect string can be used:

  ```
  SQLPLUS SCOTT/TIGER@ORAUNIX
  ```

  The definition for alias ORAUNIX is defined in the data set allocated to DDname TNSNAMES with:

  ```
  ORAUNIX=
          (DESCRIPTION=
          (ADDRESS=
          (PROTOCOL=TCP)
          (HOST=HQUNIX)
          (PORT=1533))
          (CONNECT_DATA=
          (SID=UNIX)))
  ```

Note: Oracle Corporation recommends you define all your databases in a public data set accessible by all users so they can connect to databases using aliases instead of TNS connect descriptors.

With the 9.2 release, clients open their own sockets and no longer need to direct their request through the Net address space.

Oracle Names and LDAP

Oracle clients on OS/390 can use an Oracle Names or LDAP server running on another platform to resolve connection requests. The following samples of the Oracle Net configuration file is required to make use of these services.

Name Server

```
SQLNET DD or SQLNET.ORA Definitions:
###############
# Names .........: (CONNECT_TIMEOUT = 0) -MUST- be specified
###############
NAMES.DEFAULT_DOMAIN = world
NAMES.DEFAULT_ZONE = my.domain.com
```

```
NAMES.DIRECTORY_PATH = (TNSNAMES,ONAMES, LDAP)
NAMES.PREFERRED_SERVERS =
(ADDRESS_LIST =
(DESCRIPTION =
(ADDRESS =
(PROTOCOL = TCP)
(HOST = names_host)
(Port = 1575)
)
(CONNECT_TIMEOUT = 0)
)
)
```

LDAP Server

```
LDAP DD or LDAP.ORA Definitions:
A sample LDAP.ORA file:
DEFAULT_ADMIN_CONTEXT = "c=us"
DIRECTORY_SERVERS = (hostname:389:636)
DIRECTORY_SERVER_TYPE = OID, LDAP after ONAMES
```

LDAP.ORA can be generated using the NETCA utility.

11

Migration Considerations

This chapter describes usage considerations when migrating from an earlier Oracle8*i* database with MPM or earlier OSI database to this release of the Oracle9*i* database.

The following topics are included:

Overview

Oracle9*i* database server users on OS/390 communicate with the OSDI database service address space using OS/390 cross memory services. This facility allows both data and program operation to cross address space boundaries in a secure and controlled manner.

OSDI provides a cross-memory protocol for connecting local Oracle database server clients to OSDI-managed database instances. The protocol is based on Oracle Net architecture. If migrating from MPM, this protocol replaces the SQL*Net V1-based protocols that are used by MPM. You cannot connect to an OSDI-managed server using the MPM protocol, and you cannot connect to an MPM-based Oracle instance using the OSDI protocol.

OS/390 Language Environment

In this release all Oracle clients use the IBM Language Environment for C program runtime services. This includes all Oracle tool, utilities, Access Managers and the Oracle client code used by customer written applications built using the Oracle precompilers or OCI interface.

The Oracle client code previously provided with the ORADRV load module, is now provided in LIBCLNTS. LIBCLNTS is a DLL module in CMDLOAD and must be made available to tools, utilities, Access Managers, and customer written applications either:

- by placing CMDLOAD in the STEPLIB DD concatenation in the jcl, or
- by LIBCLNTS copied into SYS1.PALIB so that one copy is shared across all uses.

The IBM Language Environment runtime library, SYS1.SCEERUN, must be made available either in the STEPLIB DD concatenation in the jcl, or by placing SYS1.SCEERUN in the LNKLSTxx member of SYS1.PARMLIB.

TOS/Batch and USS customer applications must be compiled using a Language Environment conforming compiler. See the Release Notes, "Additional Software Requirements" section for supported compilers. With migration to the use of the LIBCLNTS DLL, customer applications must be linked with a new stub as described in "Linking Your Program" in the Oracle Precompilers chapter of this book. For migration, existing applications using an OSDI version of CMDLOAD can access this version of the database as long as they continue to use the existing CMDLOAD and MSEG datasets. Existing applications built using an MPM version of CMDLOAD must be recompiled if compiled with a non-conforming Language Environment compiler and/or be relinked with the new stub.

Oracle tools and utilities now support file specifications and redirection specifications according to Language Environment syntax. See the IBM C/C++ Programming guide section on "Opening Files" and "Using Redirection Symbols" for more information. For backward compatibility Oracle Runtime file specification is supported as described in the "Oracle9i Utilities and OS/390 Files" chapter of this book. See also the appropriate chapter for the specific product being used. A previous OSDI version of tools such as SQL*Plus can access this version of the database as long as they continue to use the CMDLOAD and MESG datasets provided with that release. Prior versions of the tools provided in an MPM release will not work with this version of the database.

OS/390 Cross Memory Support

OS/390 cross memory support is provided for Oracle clients as described in the following sections.

Cross Memory Support

Oracle9*i*'s cross memory mechanism is based on Oracle Net and supports local access to Oracle database instances by its SID using the ORA@sid DD statement, local tnsnames connect string (either explicitly or using the TWO_TASK environment variable) or with the ORACLE_SID environment variable. See "Enabling OS/390 Local Client Access" on page 2-2 for details.

Applications that utilize an ORA@ssn DD statement to access an MPM-based Oracle database would require only minor JCL change to access an OSDI-managed database instance with a different SID. In other cases where other MPM specific local connection mechanisms are used, applications will need to be changed to use one of the Oracle9*i* supported mechanisms.

Support for the old SQL*Net V1-style connection strings (M:, F:, W:, and Z:) and for the CONNSTR environment variable has been removed for the Oracle9*i* and subsequent releases; therefore, a SQL*Net V1-style connection string will produce an error during the connect attempt. Any CONNSTR setting will be ignored, and other mechanisms that are lower in the precedence list will be attempted. You will need to convert scripts or applications that use either of these mechanisms to one of the others before you upgrade to Oracle9*i*.

MPM Compatibility

With MPM, a TSO or batch Oracle application has four different ways to specify a local target instance. They are, in descending order of precedence:

1. A SQL*Net V1-style connection string following an "@" character that is appended to the userid and password supplied to an Oracle connect call. This string is of the general form:

   x:*ssn*

   where *x* is the letter M, F, W, or Z and *ssn* is the 1-character to 4-character subsystem name of the target MPM. Additional protocol parameters following *ssn* such as buffer size or NOSTAX are not supported under OSDI. Note that it is the colon character that distinguishes this connection string from a Oracle Net tnsname-type specification.

2. A CONNSTR environment variable containing a connection string of the form just described in specification 1. Environment variables are supplied in a sequential file that is specified by an ORA$ENV DD statement in the batch job or is allocated to the TSO session.
3. A DD statement in a batch job (or equivalent TSO allocation) of the form:

   ```
   //ORA@ssn DD DUMMY
   ```

   where *ssn* is the 1-character to 4-character subsystem name of the target MPM Oracle instance.
4. None of the above. By default, the subsystem name MPMT is targeted.

Oracle9*i* only provides compatibility support for the ORA@ssn DD (as in method 3 above). Any applications that rely on the other mechanisms to specify a target instance will have to be changed to access an Oracle9*i* server.

These connection short-cut mechanisms all specify an MPM subsystem name. With OSDI, a single subsystem can support many Oracle instances, so the "subsystem name" that is specified by the supported mechanism is not used as such by OSDI. Instead, it is used as a SID (service identification). The SID is an identifier that is specified on the OSDI DEFINE SERVICE command. Every service must have a SID that is unique throughout the OS/390 system. Even services that are defined in different OSDI subsystems cannot have the same SID if they are on the same OS/390 system.

When you use the ORA@ssn DD described above with an OSDI-enabled client, OSDI attempts to connect to the service whose SID matches the 1-character to 4-character string that was treated as a subsystem name by MPM. OSDI SIDs can be up to 8 characters long, but if you want to exploit this client compatibility feature, then you must use a SID that matches the subsystem name of your old MPM Oracle instance, which means that the SID must be 4 characters or less. If you do not specify a SID in the DEFINE SERVICE command, then the SID name defaults to the service name. You may want to specify the SID (matching your old MPM subsystem name) so that you can use a longer, more descriptive name for the service.

Henceforth, we will refer to the DD statement mechanism for specifying an instance as "ORA@*sid*" instead of "ORA@*ssn*".

TNSNAMES Connect Descriptors

All of the mechanisms that OS/390 clients can use to specify connections, both local and remote, are listed below in descending precedence order. Aspects of each were

discussed in the prior sections as well as Chapter 2, "Using the OS/390 Database Instance".

1. A connection string following an "@" appended to the userid and password supplied on an Oracle connect request. This can be a tnsname-style name for TNSNAMES lookup, or a complete Oracle Net address string.
2. A DD statement (or TSO allocation equivalent) of the form:

   ```
   //ORA@sid  DD DUMMY
   ```

   where *sid* matches the SID of a local OSDI-managed instance
3. A TWO_TASK environment variable whose value is a tnsname-style name, or a complete Oracle Net address string.
4. An ORACLE_SID environment variable whose value matches the SID of a local OSDI-managed instance.

If none of the above four mechanisms is specified, then the client connection attempt will fail. There is no "default instance" mechanism.

Database Links

Cross-memory database links (in either direction) between MPM and OSDI instances are not supported. If you have a requirement for such distributed access, then you will need to use a network protocol (that is, TCP/IP) instead of cross-memory access. Doing this requires running both OSDI Oracle Net and TNS with distinct endpoints for the chosen protocol.

IXCF Support

IXCF is no longer supported as a separate Oracle Net for OS/390 protocol. IXCF can be supported via IBM TCP/IP in which case, Oracle Net IBM TCP/IP protocol can be used. Consult with your Oracle for OS/390 administrator regarding IXCF support with IBM TCP for your installation.

Remote Clients

All net applications now open their own tcp/ip sockets; that is, they no longer make use of the Net address for communications. Instead, each net application makes direct use of OS/390 UNIX System Services and therefore requires an OS/390 UNIX security context, also known as an OMVS segment. Each user that invokes tcp/ip

functionality must have an MVS segment if an installation is not using a default OMVS segment.

A

API Short Name Support

With this release of Oracle9*i* Enterprise Edition for OS/390, the nominal API application executable takes the form of a Program Object 3 (PM3). This appendix presents methods for producing traditional load modules if needed. The common components of these methods are the Prelinker and an alternate API stub called ORASTBS. ORASTBS is shipped as an object deck so as to be suitable for Prelinker input.

Method 1: Prelink and Link

This method is suitable for the following types of Oracle API programs; Pro*COBOL, Pro*FORTRAN, Pro*PL/I, and OCI V7 programs.

Existing Pro*C and OCI V8 programs calling API functions via 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 are passed to the Prelinker and the resultant object is then passed to the Linkage Editor.

Example:

```
//PRELINK  EXEC PGM=EDCPRLK,COND=(4,LT),PARM=''
//STEPLIB   DD DISP=SHR,
//             DSNAME=SYS1.SCEERUN
//SYSMSGS   DD DISP=SHR,
//             DSNAME=SYS1.SCEEMSGP(EDCPMSGE)
//SYSLIB    DD DISP=SHR,
//             DSNAME=ORACLE.V920.OBJLIB
//SYSIN     DD DISP=(OLD,DELETE),
//             DSN=&&LOADSET
//          DD DDNAME=SYSIN2
//SYSMOD    DD DISP=(NEW,PASS),
//             DSNAME=&&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,DYNAM=DLL'
//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. 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 and OCI V8. At precompile and/or compile time, a header file is included which maps the long names to short names. 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, are passed to the Prelinker and the resultant object is then passed to the Linkage Editor.

Example:

```
//PRECOMP  EXEC PGM=PROC,
//STEPLIB   DD DISP=SHR,
//             PARM='++/DD/SYSPARM'
//             DSN=ORACLE.V920.CMDLOAD
//ORA$LIB   DD DISP=SHR,
//             DSN=ORACLE.V920.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=ORACLE.U920.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,
//             DSNAME=SYS1.SCEERUN
//          DD DISP=SHR,
//             DSNAME=SYS1.SCBCCMP
```

```
//SYSMSGS   DD SYSOUT=*
//SYSLIB    DD DISP=SHR,
//             DSN=ORACLE.V920.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 application. For OCI V8 programs, only the compile step is needed. In either case, the object produced by the compile step, along with the alternate API stub ORASTBS, are passed to the Prelinker and the resultant object is then passed to the Linkage Editor (see Method 1 above).

Index

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W