Understanding SQL*Net®

Release 2.3

Part No. A42484–1

ORACLE®

Understanding SQL*Net, 2.3

Part No. A42484–1

Primary Authors: Larry Neumann, Sandy Venning, Laura Ferrer
Contributing Author: Shelly Dimmick Willard
Contributors: Romeo Baldeviso, George Chang, Craig Clow, Peter Chu, Carl Dellar, John Graham, Jack Haverty, Mark Hill, Mark Jarvis, Mike Kanaley, Ed Miner, Joe Pistritto, Jay Rossiter, Cyril Scott, Deb Smith, Dave Stowell, Steve Viavant, Rick Wessman, David Wong, Lelia Yin

Preface

Purpose

*Understanding SQL*Net* provides the information you need to understand and use the SQL*Net release 2.3 product. SQL*Net allows Oracle tools and applications to access, manipulate, share, and store data in Oracle databases residing on remote servers. In addition, SQL*Net enables data access between multiple database servers. This guide describes:

- an overview of SQL*Net version 2 and later
- a detailed description of the SQL*Net release 2.3 architecture
- what to consider when upgrading from SQL*Net version 1
- what to consider when upgrading from an earlier release of SQL*Net version 2
- how to use the network listener
- how to use a SQL*Net version 2 client
- important concepts about using externally identified logins
- a description of SQL*Net–related features of the Oracle7 Server
- how to use SQL*Net in conjunction with other Oracle network products

Note: This manual contains examples and figures that refer to specific machine types, network protocols, and operating systems. These references are examples of possible configurations and are not representative of all configurations.

Audience

The information in this manual is intended primarily for network or database administrators (DBAs). This guide is also for anyone who wants to understand how SQL*Net release 2.3 works.

Documentation Set

Use this guide in conjunction with the other manuals in the Oracle network products documentation set. This set consists of manuals that help you to set up an integrated, heterogeneous network and to use the applications and services provided. The documentation consists of the following manuals:

OS Doc

- Oracle operating system–specific manuals include installation instructions for the networking products on your platform, and describe the Oracle Protocol Adapters, including:
  - protocol terms and concepts
  - protocol–specific parameters

  Refer to this book for installation instructions and other information that is platform or operating system specific.

SQLNet

- *Understanding SQL*Net* (this manual)

  Describes SQL*Net release 2.3, including:
  - SQL*Net functionality
  - architecture of SQL*Net
  - new features in SQL*Net release 2.3
  - planning your SQL*Net release 2.3 network
  - using SQL*Net release 2.3

  Refer to this book to understand how SQL*Net works, and how to use it after it is installed and configured.

NetMan

- *Oracle Network Manager Administrator's Guide*

  Describes the tool that creates the configuration files for the Oracle networking products, including:
  - SQL*Net release 2.1 and later
  - Oracle MultiProtocol Interchange
  - Oracle Names
  - Oracle Native Naming Adapters
  - Secure Network Services
  - Oracle SNMP Support

Use this book when you are ready to configure the SQL*Net network.

- *Oracle MultiProtocol Interchange Administrator's Guide*


Intchg

Describes the Oracle MultiProtocol Interchange, including:

  – the purpose and function of the Interchange
  – using the Interchange to navigate connections over a network
  – client choices for accessing the Interchange
  – using the Interchange in a network

Refer to this book for information on configuration parameters and how to use the Interchange after it is installed and configured.


ONames

- *Oracle Names Administrator's Guide*

Describes the Oracle Names product, including:

  – the purpose and function of Oracle Names
  – types of network naming schemes
  – configuration requirements for Oracle Names
  – the SQL*Net Dynamic Discovery Option (DDO) for self–registering network services

Refer to this manual to learn how Oracle Names works, and how to use it after it is installed and configured.


SNS

- *Secure Network Services Administrator's Guide*

Describes the encryption, checksuming, and enhanced user authentication features that add security to the Oracle network , including:

  – an overview of the Secure Network Services product
  – the purpose and effectiveness of the security feature
  – how to enable the security features using Oracle Network Manager

Refer to this manual to learn how Secure Network Services works, and how to configure encryption, checksumming and support for enhanced authenticationwith Network Manager.

Note: Secure Network Services versions 1 and 1.1 include encryption and checksumming; version 2. also includes support for enhanced authentication.

SNMP

- *Oracle SNMP Support Reference Guide*

  Describes the Oracle SNMP Support feature, including

  – benefits, basic terms, and components of the feature, and a general overview of Oracle SNMP support for the Oracle7 Server, listener, Oracle MultiProtocol Interchange and Oracle Names.

  – an overview of the Oracle MIBs that provide Oracle SNMP Support, as well as information to help you read and interpret public and private MIB variables

  – guidelines to assist in developing management applications for Oracle products based on Oracle MIBs.

  – reading and interpreting the MIB variables that support monitoring of the Oracle7 Server, and the listener, Oracle MultiProtocol Interchange, and Oracle Names network services

  Refer to this manual to learn how SNMP support works, and how to use it after it is installed and configured.

Diagnostics

- *Oracle Network Products Troubleshooting Guide*

  Contains error and diagnostic information, including:

  – information about how to use the diagnostic features of Oracle network products. These include logging, tracing, Audit Trail, Trace Route, and the Client Status Monitor.

  – the error messages and informational messages that may occur during network processing

  Use this book for troubleshooting help.

Rel

- Release notes for this release of SQL*Net and Oracle Names.

How This Manual Is Organized

*Understanding SQL*Net* consists of six chapters and four appendices.

Chapter 1 – Introduction to SQL*Net

This chapter is an introduction to SQL*Net version 2 and how SQL*Net connections are established. It describes the general features of SQL*Net, and highlights the features new to this release.

Chapter 2 – SQL*Net Version 2 Architecture

This chapter explains the SQL*Net architecture and describes how the components of a SQL*Net network work together. It explains how

connection requests to a multi–threaded server are handled, and explains how listener load balancing works.

Chapter 3 – Network Layout and Naming Issues

This chapter outlines considerations for planning a version 2 installation. It discusses the relationships of the network products, and discusses how to plan your network for future growth and flexibility.

Chapter 4 – Migrating from an Earlier Version of SQL*Net

This chapter describes how to migrate from a SQL*Net version 1 installation or from SQL*Net version 2. It describes how to upgrade from SQL*Net version 1 and manage coexistence of two versions of SQL*Net during the migration. It also describes migration issues in a mixed environment of various SQL*Net version 2 releases.

Chapter 5 – Using SQL*Net

This chapter explains how SQL*Net is used after it has been configured. The chapter describes how to start and test the network, how to use the Listener Control Utility, how to control the clients, how to initiate connections through various tools, and how to use some distributed database features.

Chapter 6 – Configuring Oracle SNMP Support

This chapter provides a general overview of tasks to perform to configure Oracle SNMP Support for the Oracle7 Server, listener, Oracle Names, and Interchange. For detailed information on Management Information Base (MIB) variables used in the Oracle SNMP Support feature see the *Oracle SNMP Support Reference Guide.*

Appendix A – Contents of Configuration Files

This appendix describes the contents of the configuration files for SQL*Net: TNSNAMES.ORA, LISTENER.ORA, SQLNET.ORA, and PROTOCOL.ORA. (The configuration files associated with the MultiProtocol Interchange and Oracle Names are described in the *Oracle MultiProtocol Interchange Administrator's Guide* and the *Oracle Names Administrator's Guide*, respectively.)

Appendix B – Syntax Rules for Configuration Files

This appendix describes the syntax rules for configuration files. Although Oracle Network Manager creates the configuration files, this information might be useful if you need to manually edit or troubleshoot the files.

Appendix C – Common Error Messages

This appendix describes the most common causes of problems while configuring or using SQL*Net release 2.x, and discusses some of the common error messages and how to resolve them.

Appendix D – SQL*Net OPEN

This appendix describes the application programming interface (API) to SQL*Net. Using SQL*Net OPEN, programmers can develop non–database applications that take advantage of Oracle's complete set of networking products.

Related Publications

In addition to this guide and the other Oracle network product manuals, you may want to refer to the following documents published by Oracle Corporation:

- *Oracle7 Server Administrator's Guide*
- *Oracle7 Server Concepts*
- *Oracle7 Server Application Developer's Guide*
- *Oracle7 Server Distributed Systems, Volume I: Distributed Data*
- *SQL*Plus Reference Guide*
- *SQL Language Reference Manual*
- Oracle operating system–specific manuals for your platforms

Notational Conventions

The following syntax conventions are used in this manual:

`Monospace normal`	Monospace shows what displays on the computer screen or contains text you need to enter exactly as shown.
`Monospace italics`	Monospace in italics represents a variable. Substitute an appropriate value.
[ ]	Brackets enclose optional items. Do not enter the brackets.
( )	Parentheses enclose many transparent network substrate (TNS) keyword–value pairs. They must be entered as shown; for example, (KEYWORD=value).
|	A vertical bar represents a choice of two or more options. You must enter one of the options. Do not enter the vertical bar.

Punctuation	Punctuation other than brackets and vertical bars must be entered as shown.
UPPERCASE	Uppercase characters within the text represent command names, filenames, and directory names.

Note: Some operating systems are case sensitive. Although our convention is to present command names, filenames, and directory names in uppercase, that does not necessarily mean that they should be in uppercase on your platforms.

Your Comments Are Welcome

We value and appreciate your comments as an Oracle products user. As we write, revise, and evaluate our work, your opinions are the most important input we receive. At the back of this manual is a Reader's Comment Form; we encourage you to use this form to tell us what you like and dislike about this (or other) Oracle manuals. If the form is gone, or you would like to contact us, please use the following address:

Oracle Network Products Documentation Manager
Oracle Corporation
500 Oracle Parkway
Box 659410
Redwood Shores, CA 94065

Contents

List of Figures

CHAPTER

1 Introduction to SQL*Net

SQL*Net release 2.3 is Oracle Corporation's latest remote data access software. It enables both client–server and server–server communications across any network. With SQL*Net, databases and their applications can reside on different computers and communicate as peer applications. Using SQL*Net release 2.3, you can take advantage of all the features of the Oracle Server.

This chapter provides an overview of SQL*Net version 2 and later by discussing:

- database capabilities enabled by SQL*Net version 2
- features that are new to this release of SQL*Net
- the relationship of SQL*Net release 2.3 to other Oracle networking products

Functions of SQL*Net Version 2

SQL*Net version 2 provides the basis for two complementary means of distributed communications over a network: client–server and server–server computing.

Distributed Processing

In distributed or cooperative processing, clients and servers interact to resolve a single data transaction, even if the applications and databases are separate logical entities, even on separate physical machines. The transaction is distributed between the locations of the client and servers, and the client and servers must cooperate to complete the transaction.

Distributed processing allows the appropriate resources to be allocated from the appropriate machines on the network. The client, for example, can run on a user–friendly graphical workstation or desktop computer while the server resides on a machine more suited for efficient data processing. Distributed processing also allows data processing to be centralized so that many client applications can concurrently access a single server.

SQL*Net in Distributed Processing

SQL*Net is responsible for enabling communications between the cooperating partners in a distributed transaction, either client–server or server–server. Specifically, SQL*Net enables clients and servers to connect to each other, send data such as SQL statements and data responses, initiate interrupts from the client or server, and disconnect when the session is complete. During the life of the connection, SQL*Net resolves all differences between the internal data representations and / or character sets of the computers being used.

When a client or server makes a connection request, SQL*Net receives the request and, if more than one machine is involved, passes the request to its underlying layer, the transparent network substrate (TNS), to be transmitted over the appropriate communications protocol to the appropriate server. On the server, SQL*Net receives the request from TNS and passes it to the database as a network message with one or more parameters (that is, a SQL statement).

With the exception of the initial connection, both the local and remote applications generate the same requests whether they run on the same computer or are distributed across multiple computers. The initial connection differs only in that the name of the remote database must be specified by the application or user.

Figure 1 – 1 shows a local application (on the left), contrasted with a client–server application (on the right). In each case, the application is a SQL*Forms data entry screen.

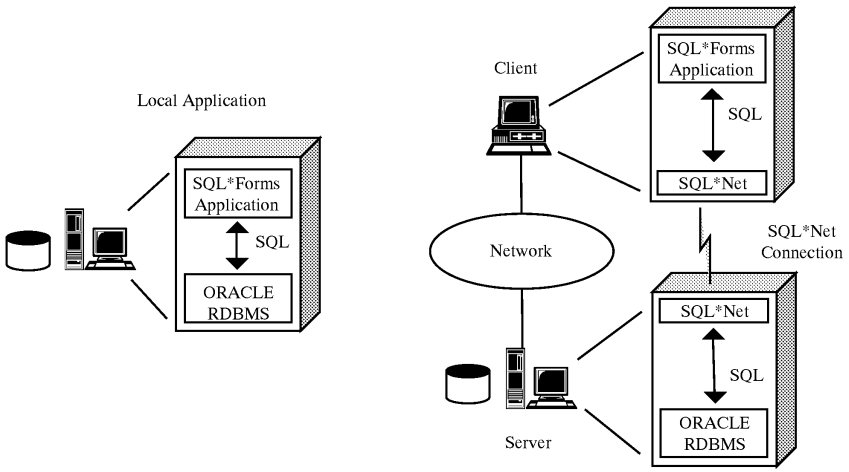

Figure 1 – 1 Local and Distributed Processing

SQL*Net Functional Benefits

SQL*Net version 2 provides the following benefits to users of networked applications:

- network transparency
- protocol independence
- media / topology independence
- heterogeneous networking
- location transparency
- diagnostic tools

Network Transparency

Oracle applications developed with a local database can be distributed across a network to access the same, or a similarly formatted, Oracle database with no changes to the application. SQL*Net is responsible for forwarding application requests for data from an Oracle client or server to a server and returning the results to the initiator of the query. From the application developer's or application user's perspective, all data interaction using SQL*Net is invisible to the user or the application. Additionally, it is possible to change the network structure beneath the application without changing the application. This quality of being invisible is known as *network transparency.*

Protocol Independence

SQL*Net provides protocol independence to its applications. An application using SQL*Net can run over any network protocol. Any application built on any computer running any protocol can be distributed without change to other computers running other protocols.

Media/Topology Independence

When SQL*Net passes control of a connection to the underlying protocol, all media and / or topologies supported by that network protocol on that platform are indirectly inherited by SQL*Net. SQL*Net allows the network protocol to use any means of data transmission, such as Ethernet, Token Ring, FDDI, or SDLC, to accomplish the low–level data link transmission between the two computers.

Heterogeneous Networking

Oracle's client–server and server–server models provide the capability for connectivity across multiple network protocols, each in a manner appropriate to its function.

Client–Server Configuration

Intchg

With SQL*Net Version 2, the client and server can belong to different *communities* connected by one or more MultiProtocol Interchanges. (For information on the MultiProtocol Interchange, see "MultiProtocol Interchange" later in this chapter and refer to the *Oracle MultiProtocol Interchange Administrator's Guide*). A community is a group of computers that can communicate using the same transport level protocol, such as TCP / IP; that is, computers that share a common protocol are said to be members of the same community. Using an Interchange as an intermediary, applications on the client and server machines can communicate in spite of having no common transport protocol. Any data exchanged in the client–server applications is forwarded through the Interchanges along the path.

Figure 1 – 2 shows a connection between a client and a server running different protocols in adjacent communities. A MultiProtocol Interchange joins the two networks. SQL*Net and an Oracle Protocol Adapter specific to Protocol A are installed on the client, while SQL*Net and an Oracle Protocol Adapter specific to Protocol B are installed on the database server. The Interchange has adapters for both Protocol A and Protocol B.

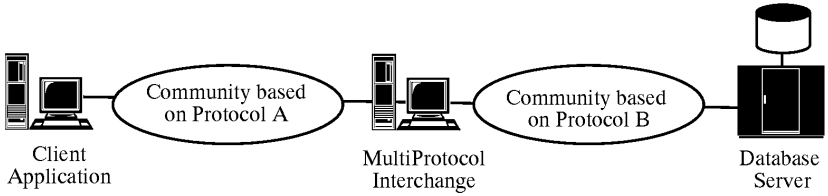

Figure 1 – 2 Heterogeneous Networking with a client–server Connection

server–server Configuration

In a server–server configuration, this same heterogeneous network capability is extended to include database-to-database communications for distributed queries and updates in Oracle7. Two types of server–server connections are possible using SQL*Net V2:

- a direct connection between two servers in the same community
- a connection between servers in different communities through one or more Interchanges

The example in Figure 1 – 3 shows both types of connections. In this example, Server 1 is a member of two communities, Community A and Community B. A client application in Community A accesses the database server (Server 1) in the same community. Server 1 determines that the transaction must be distributed further to retrieve data from tables in Server 2 and Server 3. Server 1 initiates a connection to Server 2 in Community B to which Server 1 also belongs. Server 1 also initiates a connection to Server 3 through the Interchange installed between Community B and Community C.

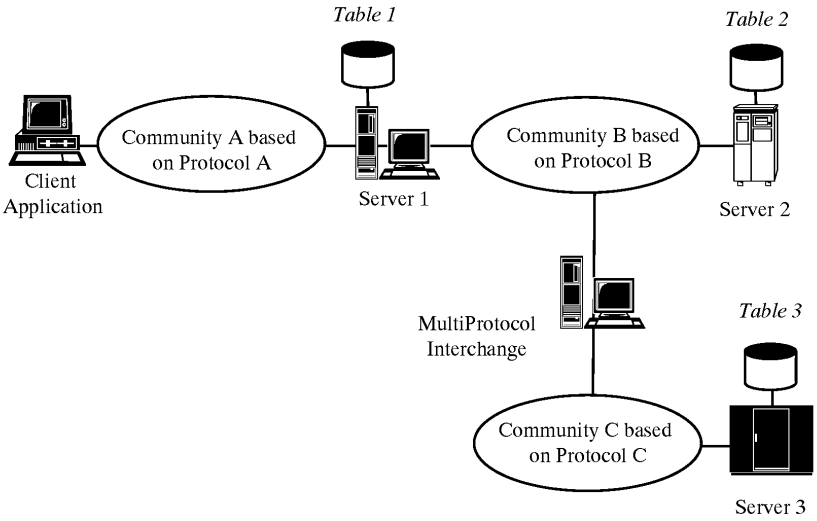

Figure 1 – 3 Heterogeneous Networking in a Distributed Database Transaction

Server 1 does not have to use an Interchange to initiate an additional request for data from Community B since it belongs to both Community A and Community B, but it has to use an Interchange to access a server in Community C.

Note that using the Interchange imposes no new restrictions on a SQL*Net connection. If the MultiProtocol Interchange is used in a client–server connection, clients have a standard peer–to–peer connection to the server despite the fact that they are in different communities. Similarly, if a server initiates a connection with another server through an Interchange using a database link, the standard database link restrictions apply.

Location Transparency

The Oracle server provides the means to make data objects such as tables in remote databases look like they are in the local database to an application developer or user of that data object. The *database link* and *synonym* database features allow the database in which they are created to identify a remote data object and make the location transparent.

This *location transparency* removes all references to the location of the data from applications when the synonym is used. Should the location of the remote table move to another machine or database, only the synonym and database link need to be updated to reference the new location; no applications changes are required. (Database links and synonyms are discussed further in Chapter 5, ”Using SQL*Net”. Also refer to *Oracle7 Server Distributed Systems, Volume I*.)

Moreover, if the database link connection is specified as a service name (or symbolic name) in the configuration file (TNSNAMES.ORA), the database links accessing the data do not have to be changed if the remote database is moved; the only update required is to the TNSNAMES.ORA file. Similarly, if Oracle Names or an Oracle Native Naming Adapter is used, only the central network definition needs to be changed.

The network definition, TNSNAMES.ORA, and other configuration files are created using Oracle Network Manager. The configuration files are described in Appendix A, "Contents of the Configuration Files". For detailed instructions on creating the network definition and the configuration files for SQL*Net and other Oracle networking products, see the *Oracle Network Manager Administrator's Guide*

NetMan

> **Note:** If you have a network in which all clients and servers use SQL*Net release 2.3, and you use Oracle Names and its Dynamic Discovery Option, minimal configuration files are necessary and you may not need to create a network definition or to use Oracle Network Manager.

Diagnostic Tools

SQL*Net release 2.3 provides a number of diagnostic tools. The diagnostic tools include the following:

- logging
- tracing
- Audit Trail
- Client Status Monitor and the SQLNET.ORA Editor
- Trace Route

Diagnostics

These tools are discussed in detail in the *Oracle Network Products Troubleshooting Guide.*

SQL*Net Release 2.3 and Other Oracle Network Products

SQL*Net release 2.3 is based on the Oracle Transparent Network Substrate (TNS) technology. It works closely with other TNS–based products to provide easy to administer, flexible, transparent communication between client and server applications. The following Oracle products are bundled with SQL*Net release 2.3:

- Oracle Network Manager
- Oracle Names

- SNMP Support
- Oracle Protocol Adapters
- MultiProtocol Interchange

The following Oracle network products are optional (that is, they are not bundled with SQL*Net release 2.3).

- Secure Network Services
- SQL*Net / DCE
- Oracle Native Naming Adapters

Please refer to Chapter 2, "SQL*Net Version 2 Architecture," for more information about TNS.

Oracle Network Manager

Oracle Network Manager release 3.1 is a tool that enables you to create the configuration files needed by Oracle networking products. It includes a graphical user interface which enables you to view the network in a hierarchical, or "tree view," structure so you can see the relationships among the network services, or in a map view which shows the network services as they relate to the user's actual network. Also provided is a context–sensitive online help system and an online walkthrough that takes you through the basic steps of configuring a network.

Oracle Network Manager enables you to validate the network so you can ensure that the resulting configuration files are free of common syntax errors and contain all required information. It includes utilities to assist in converting the configuration files of previous versions of SQL*Net to be compatible with the current version. For further information about Oracle Network Manager, including detailed instructions on how to use it to build a network, consult the *Oracle Network Manager Administrator's Guide.* Oracle Network Manager should be used only by network administrators, not individual users.

NetMan

Oracle Names

Oracle Names is transparent naming software from Oracle Corporation. It stores network names and addresses so that network components can contact one another easily without regard to their physical locations or specific configurations on the network. Access to the names and addresses is through Names Servers on the network. Included with this release of SQL*Net, Oracle Names 2.0 includes a *Dynamic Discovery*

Option (DDO), which enables servers to register themselves with well–known Name Servers. When this option is used, minimal configuration files are required. If your network uses a flat naming structure and has a limited number of servers, this option may be appropriate. Oracle Names and the Dynamic Discovery Option are described in detail in the *Oracle Names Administrator's Guide*. However, because they work so closely with SQL*Net they are discussed in this manual as well. If you do not use the Dynamic Discovery Option, you configure Oracle Names Servers by using Oracle Network Manager.

ONames

Oracle Native Naming Adapters

Oracle Native Naming Adapters enable network administrators to store Oracle service names in a native naming service already in use in their network environment. Services include Network Information Services (NIS), Distributed Computing Environment Cell Directory Service (DCE CDS), and Novell NetWare Directory Services (NDS). Using an Oracle Native Naming Adapter enables you to continue to use familiar directory services to maintain names of Oracle services. Network administrators can configure each client to use Oracle Names, TNSNAMES.ORA, or one of the native naming adapters such as NIS, to resolve Oracle service names to addresses.

OS Doc

For information on installing and configuring Oracle Native Naming Adapters, refer to your Oracle platform–specific documentation.

SNMP Support

Oracle SNMP support is provided in SQL*Net release 2.2 (and later) for the listener, Oracle7 Server, Oracle Names, and MultiProtocol Interchange. SNMP support allows a database to be remotely monitored by any SNMP–capable management software in a TCP / IP network. SNMP (the Simple Network Management Protocol) is a de facto standard underlying many popular network management systems such as Hewlett Packard's OpenView, Novell's Network Management System, IBM's NetView / 6000, and Sun Solstice. It enables Oracle products such as the SQL*Net network listener, Oracle7 Server, MultiProtocol Interchange, and Oracle Names to be located, identified, and monitored by a management station running at one or more centrally located nodes.

NetMan

For information on configuring SNMP support for the listener, Oracle7 Server, Oracle Names, and MultiProtocol Interchange, see the *Oracle Network Manager Administrator's Guide* and your Oracle platform–specific documentation. Also see Chapter 6, "Configuring

SNMP

SNMP Support," in this guide for a general overview of configuring SNMP support for Oracle services. For detailed information about SNMP Support and the information it provides, see the *Oracle SNMP Support Reference Guide*.

Oracle Protocol Adapters

Oracle Protocol Adapters enable SQL*Net to establish connections over specific protocols or networks. Each machine on a network requires at least one protocol adapter, and additional adapters may be required if you use multiple protocols in your network. For example, a machine connecting a TCP/IP network and an SPX/IPX network would require a protocol adapter for each network. Also, the same machine would likely be running a MultiProtocol Interchange to enable communication between the dissimilar networks.

On some platforms, a single Oracle Protocol Adapter can operate on hundreds of different network interface boards, which means that you can deploy applications in any networking environment, including Ethernet, Token–Ring, FDDI (Fiber Distributed Data Interface), ATM (Asynchronous Transfer Mode), and wireless.

MultiProtocol Interchange

Intchg

The Oracle MultiProtocol Interchange works with SQL*Net Version 2 to provide transparent communication between disparate communities. The product is described in detail in the *MultiProtocol Interchange Administrator's Guide*. However, because it works so closely with SQL*Net Version 2, it is mentioned frequently in this manual as well. The MultiProtocol Interchange is configured using Oracle Network Manager.

Secure Network Services

Oracle Secure Network Services is an optional product that works with SQL*Net release 2.1.4 and later. It enables SQL*Net and related products to use network data encryption and checksumming so that data cannot be read or altered. It protects data from unauthorized viewing by using the RSA Data Security RC4™ or the Data Encryption Standard (DES) encryption algorithm. To ensure that data has not been modified, deleted, or replayed during transmission, Secure Network Services optionally generates a cryptographically secure message digest and includes it with each packet sent across the network.

Secure Network Services is supported by the MultiProtocol Interchange which means that clients and servers using different protocols can securely transfer data across network protocol boundaries. For example, clients using LAN protocols such as NetWare (SPX / IPX) can securely share data with large servers using different protocols such as LU6.2, TCP / IP, or DECnet.

Secure Network Services version 2 includes enhanced user authentication services such as support for single sign–on by way of Kerberos–based authentication servers and smartcard authentication based on Security Dynamics, Inc. SecurID card. These authentication services enhance the existing security facilities of Oracle7 such as access control logon, roles, and auditing by providing reliable user identification. No changes to applications are required. Secure Network Services works over all protocols, operating systems, and name services.

These services are available to most products that implement SQL*Net, including the Oracle7 Server, Developer 2000 tools, and any other Oracle or third–party product that supports SQL*Net.

SQL*Net/DCE

SQL*Net / DCE is an optional product that works with SQL*Net 2.1.6 and later. It enables users to transparently use Oracle tools and applications to access Oracle7 Servers in a DCE environment. It provides authenticated RPC (Remote Procedure Call) as the transport mechanism, which enables multi–vendor interoperability. The DCE security service enables a user logged onto DCE to securely access any Oracle application without having to specify a username or password. This is sometimes referred to as ”external authentication,” formerly referred to as ”OPS$ support”.

SQL*Net / DCE also provides support for DCE Cell Directory Service (CDS), which allows Oracle7 services to be transparently accessed throughout the DCE environment. Users can connect to Oracle database servers in a DCE environment using familiar Oracle service names. Oracle service names can be managed from a central location with standard DCE tools. For information on configuring SQL*Net / DCE, see the *SQL*Net/DCE Administrator's Guide* and your Oracle platform–specific documentation.

OS Doc

SQL*Net's New Features

This release of SQL*Net provides a number of new features and enhancements over previous releases. These enhancements include features that

- improve performance
- make SQL*Net easier to configure and use
- provide increased troubleshooting capability
- make SQL*Net networks available to non–database applications.

Performance Improvements

With SQL*Net release 2.3 you can choose to have multiple listeners for a single database and multiple listeners listening for more than one equivalent database. *Listener load balancing* enables you to distribute connection requests for a database among a number of listeners. With connections distributed among a number of listeners, no single listener is likely to be overburdened, and connection time will be faster. You can also have a many–to–many relationship between listeners and equivalent database instances. For more information about listener load balancing, see Chapter 2.

Ease of Use Enhancements

A number of new features make setting up and maintaining a SQL*Net release 2.3 network easier.

- Oracle Names version 2 with the Dynamic Discovery Option

  If you create a new network using Oracle Names version 2 and the Dynamic Discovery Option, you may not need to use Oracle Network Manager to create a network definition or extensive configuration files. Even if you add Oracle Names 2.0 and the Dynamic Discovery Option to an existing network, you can reduce the number of configuration files needed and the amount of effort required to add new clients and servers to the network. This option is suitable if the network has a flat naming structure, a single transport protocol, and a limited number of servers. For further information, see Chapter 3 in this manual and the *Oracle Names Administrator's Guide*.

ONames

- SQLNET.ORA Editor

  Using the new SQLNET.ORA Editor, which you access through the Client Status Monitor, you can edit individual client configuration files without using Oracle Network Manager. This tool makes it easy to turn client tracing off and on and to modify other optional parameters. For further information, see Chapter 5

Diagnostics

in this manual and Chapter 4 in the *Oracle Network Products Troubleshooting Guide.*

- New Listener Control Utility commands

  Through the Listener Control Utility (LSNRCTL) you can set a number of listener parameters using the SET command, or display the parameters using the SHOW command. You can also change the listener password using LSNRCTL. For further information, see Chapter 5 in this manual.

Troubleshooting Features

New diagnostic tools have been added to this release of SQL*Net.

- Audit Trail

  The new Audit Trail feature can be valuable to a network or database administrator, or anyone responsible for monitoring user activity. This feature adds a block of text to existing listener log files every time a connection is attempted by a client. For further information, see Chapter 2 in the *Oracle Network Products Troubleshooting Guide.*

Diagnostics

- Trace Route

  Trace Route (*TrcRoute*) is a new utility that allows administrators to discover what path or route a connection is taking from the client to the server. TrcRoute travels as a special type of connect packet. For more information about TrcRoute, see Chapter 5 in the *Oracle Network Products Troubleshooting Guide.*

Diagnostics

- Client Registration

  Client registration enables you to provide information about a client that is included in the Audit Trail when the client makes service requests. This registration information is captured in the audit trail and can be loaded into database tables for tracking purposes. It may be useful in compiling usage or billing statistics.

- Client Status Monitor

  The Client Status Monitor is a utility that provides a snapshot of the client's current configuration. Diagnosing connection problems and providing information to Oracle Customer Support are made easier using this tool. The Client Status Monitor also provides access to the SQLNET.ORA Editor, described earlier in this section. For more information about the Client Status Monitor, see Chapter 5 in this manual, and Chapter 4 in the *Oracle Network Products Troubleshooting Guide.*

Diagnostics

Open API	The SQL*Net OPEN API enables developers to create non–database applications that make use of an existing SQL*Net network. Refer to Appendix D for further information.

CHAPTER

2 SQL*Net Version 2 Architecture

SQL*Net version 2 uses the Transparent Network Substrate (TNS) and industry–standard network protocols to connect a client to a server and establish an Oracle session.

The next few sections provide a discussion of SQL*Net and the role it plays in distributed systems. The following architectural concepts are discussed:

- Transparent Network Substrate
- SQL*Net's communication role
- distributed processing with SQL*Net
- SQL*Net operations
- SQL*Net and the network listener

Transparent Network Substrate (TNS)

Forming the basis for Oracle networking products, the Transparent Network Substrate (TNS) enables Oracle to provide a network of applications above all existing networks of computers. With TNS, peer–to–peer application connectivity is possible where no direct machine–level connectivity exists.In *peer–to–peer* architecture, two or more nodes can communicate with each other directly, without the need for any intermediary devices. In a peer–to–peer system, a node can be both a client and a server.

TNS provides two key features to a TNS–based network product and, in turn, any application built using TNS:

- a single, common interface to all industry–standard protocols
- the ability to connect to applications in physically separate networks through one or more MultiProtocol Interchanges

TNS is the foundation component of all current and planned network products from Oracle. Today, TNS networks connect Oracle clients and servers through SQL*Net version 2. In the future, Oracle Corporation will provide additional TNS–based application connectivity tools.

SQL*Net's Communication Role

Communication between client and server proceeds in a stack–like fashion with corresponding levels communicating in a peer relationship. The logical exchange unit at each layer of the stack conveys the level of generalization employed at that level. The Oracle client and server exchange SQL statements and data rows. At the UPI / OPI (User / Oracle Program Interface) layers, these exchanges translate into series of calls to SQL routines such as logon, parse, execute, and fetch. In a distributed transaction, SQL*Net is responsible for sending information across various networks on behalf of a client application or database server. In such a configuration, there are commonly two types of computers acting as the client and server.

Two–Task Common (described on page 2 – 4) ensures that all differences between clients and servers, such as internal datatype representations or NLS character sets, are resolved, allowing the client and server to communicate transparently. SQL*Net relays all communication tasks to TNS through its common entry points. SQL*Net is unaffected by the specific communication mechanism used underneath TNS (for example, TCP / IP, DECnet, shared memory, and so on). The SQL*Net layer handles these calls as a series of Oracle send / receive messages, and TNS in turn processes the packets over the network. The network protocol, not provided by Oracle (typically provided with each particular platform by its vendor), supplies a reliable means of communication between the two tasks.

Components Involved in Distributed Processing

Several software components are involved in completing a distributed transaction, whether it is a client–server or server–server transaction. Figure 2 – 1 shows the components of a client–server session. These components are described in the following sections.

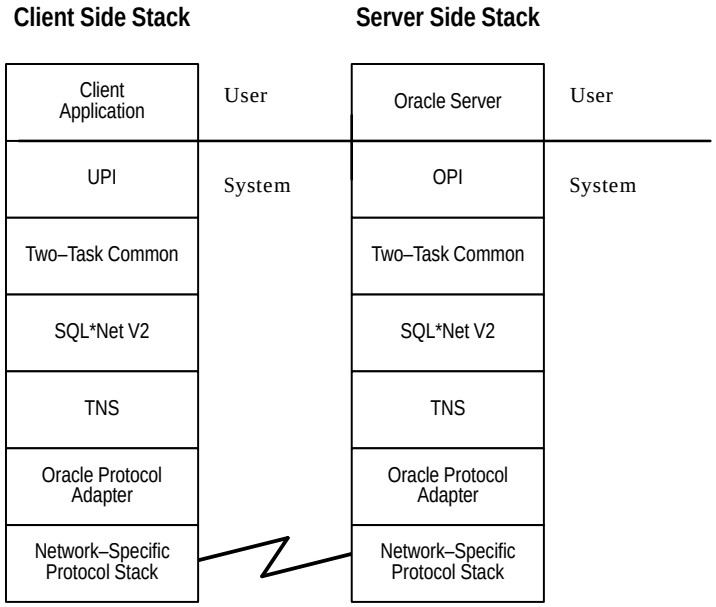

Figure 2 – 1 Oracle Client–Server Components

Client–Side Interaction

The following paragraphs discuss the components of the client–server transaction process, beginning with the client application and concluding with the Oracle Server.

Client Application

The client application provides all user–oriented activities, such as character or graphical user display, screen control, data presentation, application flow, and other application specifics. The application identifies any SQL database operations to send to the server database and passes them through the User Program Interface (UPI).

User Program Interface (UPI)

The UPI code contains all information required to initiate a SQL dialogue between the client and the server. It defines calls to the server to:

- parse SQL statements for syntax validation
- open a cursor for the SQL statement
- bind client application variables into the server shared memory

- describe the contents of the fields being returned based on the values in the server's data dictionary
- execute SQL statements within the cursor memory space
- fetch one or more rows of data into the client application
- close the cursor

The client application uses some combination of these calls to request activity within the server. Often, all UPI calls can be combined into a single message to the server, or they may be processed one at a time through multiple messages to the server, depending on the nature of the client application. Oracle products attempt to minimize the number of messages sent to the server by combining many UPI calls into a single message to the server. When a call is performed, control is passed to SQL*Net to establish the connection or transmit the request to the server.

Two–Task Common

Two–Task Common provides character set and data type conversion between different character sets or formats between client and server. This layer is optimized to perform conversion only when required on a per connection basis.

At the time of initial connection, SQL*Net version 2 is responsible for evaluating differences in internal data and character set representations and determining whether conversions are required for the two computers to communicate.

SQL*Net

The role of SQL*Net is to establish and maintain a connection between the client application and the server and exchange messages between them. The network listener receives connection requests for a particular database and passes control to the server.

Transparent Network Substrate (TNS)

TNS receives requests from network applications, in this case SQL*Net, and settles all generic machine–level connectivity issues, such as:

- the location of the server or destination (open**,** close functions)
- whether one or more MultiProtocol Interchanges will be involved in the connection (open**,** close functions)
- how to handle interrupts between client and server based on the capabilities of each (send, receive functions)

The generic set of TNS functions (open, close, send, receive) passes control to an Oracle Protocol Adapter to make a protocol–specific call.

SNS

Additionally, TNS optionally provides encryption and sequenced cryptographic message digests to protect data in transit. See the*Secure Network Services Administrator's Guide* for more information.

Oracle Protocol Adapter

The Oracle Protocol Adapters are responsible for mapping TNS functionality to industry–standard protocols used in the client–server connection. Each adapter is responsible for mapping the equivalent functions between TNS and a specific protocol.

Network–Specific Protocols

All Oracle software in the client–server connection process requires an existing network protocol stack to make the machine–level connection between the two machines. The network protocol is responsible only for getting the data from the client machine to the server machine, at which point the data is passed to the server–side Oracle Protocol Adapter.

Server–Side Interaction

Going up the process stack on the server side is the reverse of what occurred on the way down the client side. See the right side of Figure 2 – 1.

The one operation unique to the server side is the act of receiving the initial connection. The server has a process (the network listener) that regularly checks for incoming connections and evaluates their destination.

The network listener is a process on a server that listens for connection requests for one or more databases on one or more protocols. It is discussed in "SQL*Net and the Network Listener" on page 2 – 9. Based on the Oracle Server ID (SID) specified, the connection is passed to the Oracle Server.

The components above SQL*Net, the OPI and the Oracle Server, are different from those on the client side.

Oracle Program Interface (OPI)

The OPI has a complementary function to that of the UPI. It is responsible for responding to each of the possible messages sent by the UPI. For example, a UPI request to fetch 25 rows would have an OPI response to return the 25 rows once they have been fetched.

Oracle Server

The Oracle Server side of the connection is responsible for receiving dialog requests from the client UPI code and resolving SQL statements on behalf of the client application. Once received, a request is processed and the resulting data is passed to the OPI for responses to be formatted and returned to the client application.

Server–to–Server Interaction

When two servers are communicating to complete a distributed transaction, the process and dialogues are the same as in the client–server scenario, except that there is no client application. See Chapter 5, "Distributed Updates" in *Oracle7 Server Distributed Systems, Volume I* for more information. The server has its own version of UPI, called NPI. The NPI interface can perform all of the functions that the UPI does for clients, allowing a coordinating server to construct SQL

requests for additional servers. Figure 2–2 shows a server–to–server connection and all associated layers.

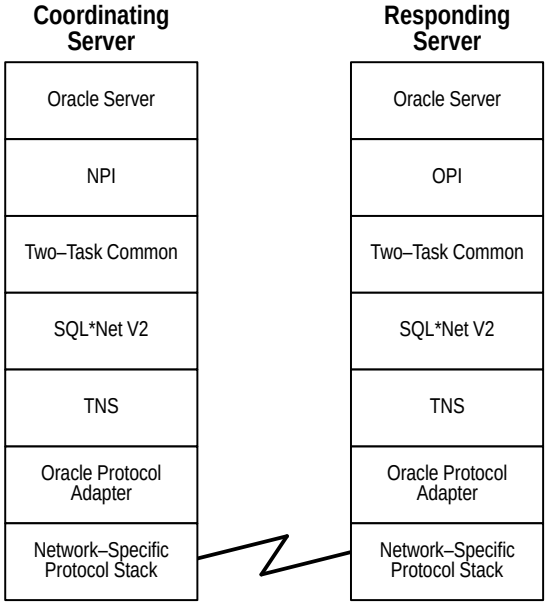

Figure 2 – 2 Oracle Server–Server Components

SQL*Net Operations

SQL*Net provides functions, described in the following sections, that belong to the following classifications:

- connect operations
- data operations
- exception operations

All the functions work with tools and databases that use SQL*Net for distributed processing, although none of them is visible to the user.

> **Note:** The information contained in the following summary is for the benefit of the network administrator, who needs to understand what role SQL*Net version 2 plays within the network.

Connect Operations

SQL*Net supports two basic connect operations:

- connect to server (open)

- disconnect from server (close)

Connecting to Servers

The connect operation is initiated during any standard database login between the client application and the server, with information such as the client machine name and username being passed to the remote machine. This information is required to support externally identified logins.

A client application initiates a request for a connection to a remote database (or other network service) by providing a short name for its desired destination. That short name, called a *service name*, is mapped to a network address contained in a *connect descriptor* stored in the network configuration file TNSNAMES.ORA, in a database for use by Oracle Names, or in native naming service such as NIS or DCE CDS.

> **Note:** If the network includes Oracle Names, the service names and associated connect descriptors are stored in a database that is accessed by the Names Servers, and the TNSNAMES.ORA file is not needed. Similarly, if an Oracle Native Naming Adapter such as NIS or DCE CDS is being used, this information will be stored and retrieved from that native name service.

Disconnecting from Servers

Requests to disconnect from the server can be initiated in the following ways:

- user–initiated disconnect
- additional connection request
- abnormal connection termination
- timer–initiated disconnect

User–Initiated Disconnect A user can request a disconnection from the server when a client–server transaction completes. A server can also disconnect from a second server when all server–server data transfers have been completed, and no need for the link remains (the simplest case).

Additional Connection Request If a client application is connected to a server and requires access to another user account on the same server or on another server, most Oracle tools will first disconnect the application from the server to which it is currently connected. Once the disconnection is completed, a connection request to the new user account on the appropriate server is initiated.

Abnormal Connection Termination Occasionally, one of the components below SQL*Net will be disconnected or will abort communications and SQL*Net will not be immediately informed.

During the next SQL*Net data operation, the TNS module will recognize the failure and give SQL*Net a notice to clean up client and server operations, effectively disconnecting the current operation.

Timer Initiated Disconnect or Dead Connection Detection (SQL*Net release 2.1 and later only). Dead connection detection is a feature that allows SQL*Net to identify connections that have been left hanging by the abnormal termination of a client. On a connection with Dead Connection Detection enabled, a small probe packet is sent from server to client at a user–defined interval (usually several minutes). If the connection is invalid (usually due to the client process or machine being unreachable), the connection will be closed when an error is generated by the send operation, and the server process will exit.

This feature minimizes the waste of resources by connections that are no longer valid. It also automatically forces a database rollback of uncommitted transactions and locks held by the user of the broken connection.

Data Operations

SQL*Net supports four sets of client–server data operations:

- send data synchronously
- receive data synchronously
- send data asynchronously
- receive data asynchronously

The concept of sending and receiving data between client and server on behalf of the UPI and OPI is relatively straightforward. A SQL dialogue request is forwarded from the UPI using a send request in SQL*Net. On the server side, SQL*Net processes a receive request and passes the data to the database. The opposite occurs in the return trip from the server.

The basic send and receive requests are synchronous. That is, when the client initiates a request, it waits for the server to respond with the answer. It can then issue an additional request.

SQL*Net version 2 adds the capability to send and receive data requests asynchronously. This capability was added to support the Oracle7 multi–threaded server, which requires asynchronous calls to service incoming requests from multiple clients.

Exception Operations

SQL*Net supports three exception operations:

- initiate a break over the TNS connection
- reset a connection for synchronization after a break
- test the condition of the connection for incoming break

Of these three operations, only the initiation of a break can be controlled by the user. When the user presses the Interrupt key [Ctrl–C] on some machines), the application calls this function. Additionally, the database can initiate a break to the client if an abnormal operation occurs, such as during an attempt to load a row of invalid data using SQL*Loader.

The other two exception operations are internal to some products using SQL*Net to resolve network timing issues. SQL*Net can initiate a test of the communication channel, for example, to see if new data has arrived. The reset function is used to resolve abnormal states, such as getting the connection back in synchronization after a break operation has occurred.

SQL*Net and the Network Listener

TNS includes a protocol–independent application listener that receives connections on behalf of any TNS application, over any underlying protocol. Referred to as a *network listener*, it runs as a single process or task and can service the needs of all TNS applications over all protocols available on a machine.

SQL*Net version 2, as a TNS–based product, uses the network listener on a server to receive incoming connections from SQL*Net clients. The network listener listens for SQL*Net connections on a specific port or socket, which is defined in the ADDRESS portion of the connect descriptor. A service may have more than one listener if needed. For more information about using multiple listeners for a database server, see ”Listener Load Balancing” later in this chapter.

Network Listener and Native Listeners

The network listener is available for all standard transport protocols supported by TNS. In addition, there are protocols that have application generic listeners or connection acceptance methods, such as DECnet and APPC/LU6.2, that may receive TNS connections.

OS Doc

Additional Information: For information on SQL*Net version 2 connections with a native connection acceptance method, see the Oracle operating system–specific documentation for that protocol and platform.

Prestarted Dedicated Server Processes

SQL*Net release 2.1 and later provides the option of automatically creating dedicated server processes. With this option, when the listener starts, it creates Oracle server processes which are then available to service incoming connection requests. These processes may last for the

life of the listener, and they can be reused by subsequent connection requests.

> **Note:** Prestarted dedicated servers require SQL*Net release 2.1 or later, and require Oracle7 Server release 7.1 or later.

Prestarted dedicated server processes reduce connect time by eliminating the need to create a dedicated server process for each new connection request as it comes to the listener. They also provide better use of allocated memory and system resources by recycling server processes for use by other connections without having to shut down and recreate a server. The use of prestarted dedicated server processes is particularly useful in systems where the Oracle7 multi–threaded server is unsupported, or where the creation of a new server process is slow and resource–intensive.

Figure 2 – 3 shows the role of the network listener in a SQL*Net connection to a server connected to two communities.

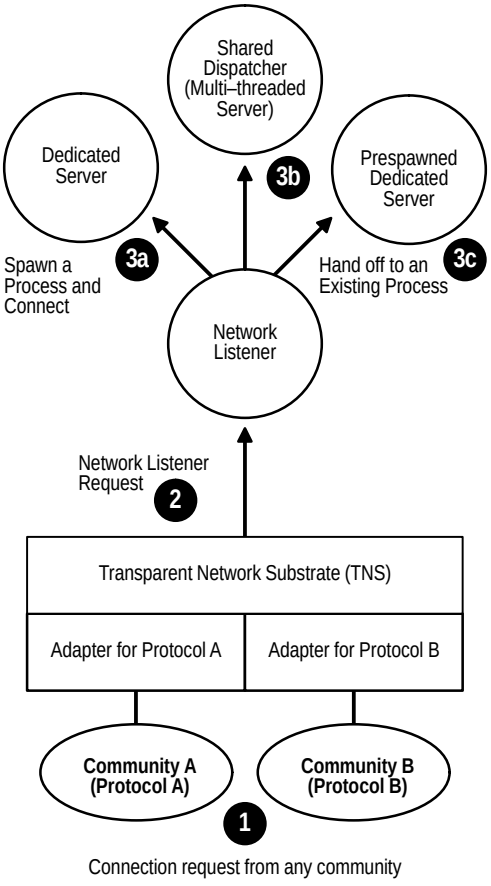

Figure 2 – 3 Network Listener in a SQL*Net Connection

The steps involved in establishing a connection (as shown in Figure 2 – 3) are:

Step 1. A connection request is made by any client in the TNS network and arrives through one of the communities to which the listener is attached.

Step 2. The network listener identifies that a connection request has arrived in one of its communities.

Step 3. **a.** The network listener spawns a dedicated server process and passes control of the incoming connection to it, or,
b. the address of a shared dispatcher process (multi–threaded server) is provided, and the incoming connection is directed to it, or,

c. the incoming connection is redirected to one of the prespawned dedicated server processes.

At the completion of a connection, the network listener continues to listen for additional incoming connections.

How SQL*Net Establishes Connections to a Prestarted Dedicated Server

Prestarted (commonly referred to as ”prespawned”) Oracle7 Servers are server processes that are prestarted by the listener before any incoming connection request. They improve the time it takes to establish a connection on servers where the multi–threaded server is not used or not supported on a given machine. Their use in a heavily loaded distributed system can be beneficial.

Set the following parameters using Oracle Network Manager. They control how the server is prespawned.

PRESPAWN_MAX	The maximum number of prespawned servers the listener creates. This value should be a large number and at least the sum of the POOL_SIZE for each protocol.
POOL_SIZE	The number of unused prespawned server processes for the listener to maintain on the selected protocol. The number must be greater than zero, but no larger than the PRESPAWN_MAX value. Set this value to the average expected number of connections at any given time.
TIMEOUT	The time that an inactive server process should wait for the next connection before it shuts down. This parameter is used to prevent server processes from being immediately shut down after a client disconnects. For greatest efficiency, provide a short time value for this parameter.

You can set specific prespawned server parameters for each SID. Thus, systems with heavy use can be tailored to accommodate the larger number of connection requests by setting PRESPAWNED_MAX and POOL_SIZE to large values. Similarly, when systems require mostly shared connections, the number of prestarted servers can be set to a low value.

Following is the sequence of events that occur when you are using prestarted servers to service client connection requests:

1. The listener is started and listens on the addresses preconfigured in LISTENER.ORA, created by the network administrator using Network Manager.
2. The listener then spawns a series of server processes until it reaches the POOL_SIZE for each SID defined in LISTENER.ORA.
3. Each spawned server process performs a wildcard listen and provides the listener with the wildcard address that it is listening on. The listener initially marks all prestarted servers as idle.
4. The client calls the preconfigured well–known address of the listener.
5. The listener receives the connection request, performs the connection handshake and determines if the client is allowed to connect. If not, the listener refuses the connection and then resumes at step 9.
6. The listener issues a redirect message to the client containing one of the wildcard listen addresses of the prespawned servers. The listener then logs that server as active.
7. The client dissolves the connection to the listener and then establishes a connection to the prespawned server using the address provided in the redirect message.
8. The listener spawns another server to replace the active prespawned server (provided the PRESPAWN_MAX value is greater than the number of prespawned server processes active and idle).
9. The listener continues listening for incoming connections.

The above sequence of events continues until the PRESPAWN_MAX is reached, at which point the listener will cease spawning new servers.

When clients disconnect, the prespawned server associated with the client is returned to the idle pool. It then waits the length of time defined in the TIMEOUT parameter to be assigned to another client. If no client is handed to the prespawned server before TIMEOUT expires, the prespawned server shuts itself down.

NetMan

See the *Oracle Network Manager Administrator's Guide* for more information on configuring this feature.

How SQL*Net Establishes Connections to a Multi–Threaded Server

The multi–threaded server enables many clients to connect to the same server without the need for dedicated server processes for each client. Using the multi–threaded server enables you to minimize the memory and processing resources needed on the server side as the number of connections to the database increases.

The sequence of events that occurs with the Oracle7 multi–threaded server is as follows:

- The listener and the multi–threaded server start up.
- Clients connect to the Oracle7 multi–threaded server.

What Happens When an MTS and Listener Are Started

During initial startup of the Oracle7 multi–threaded server and the listener, the following sequence occurs:

1. The listener starts and listens for the addresses preconfigured in the LISTENER.ORA file. (The network administrator creates this file with Network Manager, or, if Oracle Names 2.0 and the Dynamic Discovery Option are used, the LISTENER.ORA file is created during installation.)
2. The DBA starts the Oracle7 database. Dispatchers start according to the configuration parameters in the initialization parameter file. Dispatchers each use one particular protocol. There may be many, on assorted protocols. Each dispatcher performs a wildcard listen for the protocol assigned to it.

   Note: A *wildcard listen* is where the server process listens, but informs the underlying protocol stack (or operating system in the case of the IPC Protocol Adapter) that it has no preference as to what address it listens for other than specifying the protocol on which it wishes to perform the operation. As a result, many operating systems will choose a free listening address and automatically assign this to the requesting server process.
3. Each dispatcher calls the listener using the protocol it is assigned, at the address defined in the MTS_LISTEN_ADDRESS in the initialization parameter file. There may be more than one such address if multiple listeners are used.

   Note: If step 2 is performed before step 1, the dispatchers will be unable to contact the listener in step 3. If this occurs, each dispatcher loops and attempts to reconnect to the listener every 60 seconds. Meanwhile, incoming connection requests will be handled through other means (prespawned dedicated or newly spawned dedicated server processes).

The listener and the Oracle7 multi–threaded server should be ready for incoming connections at this point. You can check which dispatchers have registered with the listener by typing

```
lsnrctl services listener_name
```

How a Multi–Threaded Server Connection Request Is Handled

The following is how a multi–threaded server connection request is handled:

1. The client calls the address of the listener for the desired database. If there is more than one listener, the client randomizes its calls among them.
2. The listener receives the connection request, performs the connection handshake and determines if the client is allowed to connect (by checking the list of SIDs it listens for), at which point it continues with step 6. If not, the listener refuses the connection and then resumes at step 6.
3. The listener issues a redirect message back to the client containing the address of the least–called dispatcher that is listening on the protocol used by the client.
4. The client closes the connection to the listener and then establishes a new connection to the dispatcher, using the address provided by the listener in the redirect message.
5. The listener and dispatcher perform a short handshake to update each other of the presence of a new connection. This is so that the listener can load balance connections between dispatchers running on the same protocol.
6. The listener resumes listening for incoming connections.

When an Oracle7 Server has been configured as a multi–threaded server, incoming connections are always routed to the dispatcher unless the connection request specifically requests a dedicated server (by having SERVER=DEDICATED in the CONNECT_DATA portion of the connect descriptor) or no dispatchers are available. To create this parameter, create an alias for the database using Network Manager. (An *alias* is an alternative name that is mapped to a service name.) Use Network Manager to make that alias a dedicated server. See Chapter 5 in the *Oracle Network Manager Administrator's Guide* for more information.

NetMan

Listener Load Balancing

The listener load balancing feature provides for the distribution of connections among listeners for Oracle7 Servers. By having connections distributed among a number of listeners, no single listener is likely to be overburdened, and connection time will be faster.

> **Note:** You may have a many–to–many relationship between listeners and equivalent database instances.

Once randomly selected, the listeners behave as they have in previous releases, distributing connection requests to the dispatchers on the basis of their load (for multi–threaded servers) or to dedicated servers.

How Listener Load Balancing Works

There can be multiple listeners for a single database or multiple listeners for two or more equivalent databases.

Multiple Listeners for a Single Database

You can have more than one listener for a single database of any kind. The listeners for a dedicated server must be on the same node as the server. Listeners for a multi–threaded server can be distributed on different nodes; the dispatchers are able to register with listeners across nodes.

Suppose there were listeners on two different nodes that were listening for a single multi–threaded server, as shown in Figure 2 – 4.

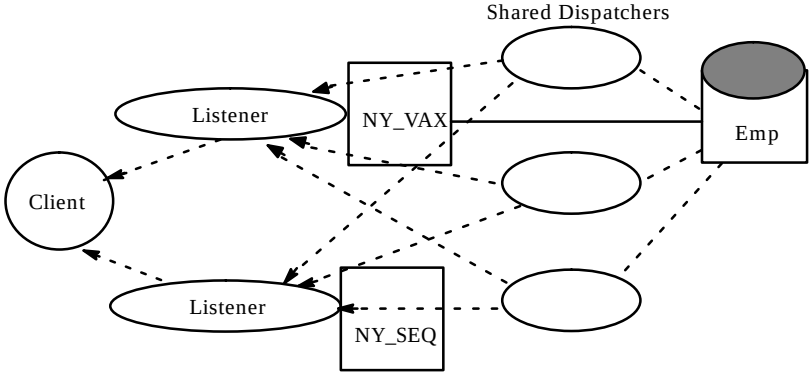

Figure 2 – 4 Multiple Listeners for a Multi–Threaded Server

All dispatchers register with all the listeners listed in the database parameter file, and the client randomizes between the listeners. The listener passes the address of the least–used dispatcher to the client, and the client connects using that dispatcher.

Multiple Listeners for Equivalent Databases

If there is a set of databases configured to provide equivalent service (such as a replicated database) on the network, you can use listener load balancing among several listeners that all listen for more than one instance of the database. An example of this is shown in Figure 2 – 5.

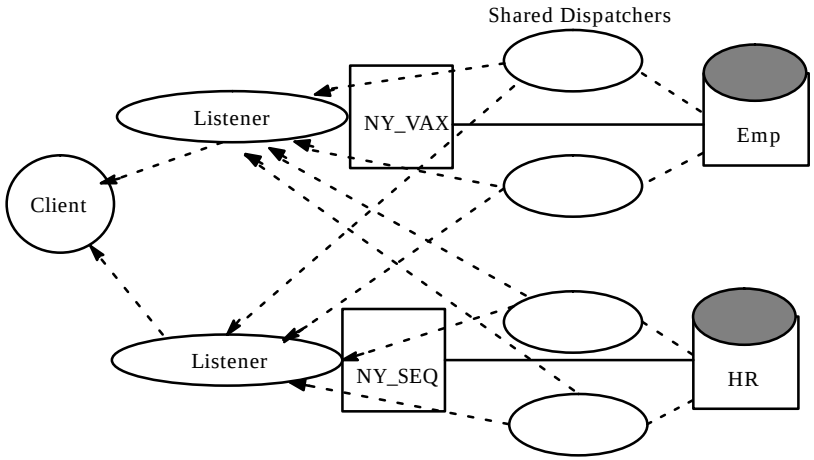

Figure 2 – 5 Multiple Listeners for Equivalent Databases

In this figure, both listeners listen for two database instances. The client randomizes between the listeners. The listener passes the address of the least–used dispatcher to the client, regardless of whether the dispatcher is for Emp or HR, because they are virtually identical.

> **Note:** Oracle Parallel Servers have their own method of listener load balancing. See your Oracle Parallel Server and platform–specific documentation on how to configure parallel servers. Use the listener load balancing method you prefer.

Listener load balancing may be particularly useful in replicated environments, where a particular group of users may have a schema that's the same across different databases, even though other things on the database may be different. In such a case, you could allow the users to update any of the replicas they happened to get connected to, and synchronize them later.

How Connections Are Made

The dispatcher for each multi–threaded server registers with all listeners designated in the MTS_LISTENER_ADDRESS parameters in the database parameter file. When a connection request is made, the client interface code randomizes among listeners within the same community. The listener redirects the client connection request to the least–used dispatcher, which connects the client to a shared server process.

Similarly, the clients that want to connect to a dedicated server randomize their connection requests among the listeners on the same protocol for that server, as listed in the TNSNAMES.ORA file, or stored in the Names Server cache. All listeners for a dedicated server must be on the same node. The randomly selected listener spawns a server process and hands off the incoming connection to the server.

> **Note:** You cannot have listener load balancing with prestarted dedicated servers, because a prestarted dedicated server process registers only with the listener that started it.

Configuring Listener Load Balancing

For information about how to configure listener load balancing, see ”Configuring Listener Load Balancing” in Appendix A.

CHAPTER

3

Network Layout and Naming Issues

This chapter describes some of the decisions you need to make about the structure of the network. Read this chapter carefully before using Oracle Network Manager to configure the network. This chapter includes discussions of the following topics:

- deciding on network layout
- deciding whether to use Oracle Names and the Dynamic discovery Option (DDO)
- naming the components
- naming considerations and concepts
- preparing to use the Oracle Network Manager

Suggestion: When planning your SQL*Net network, think about future needs as well as present requirements. Select a layout that is flexible and expandable. If you foresee your network growing, select computers that have the capacity to handle additional connections. When naming the components in your system, think about how your naming conventions can be extended to handle future components.

Plan the Network Layout

It is a good idea to draw a picture of your network layout as you make decisions about its composition. Especially if your network includes multiple communities, Interchanges and Names Servers, it is much easier to understand and modify if you have a diagram. Two types of diagrams are useful:

- physical diagrams
- logical diagrams

Physical diagrams show every component in a network, including the physical connections among them. A physical diagram can help show what pieces are required and demonstrate the connections between components.

Logical diagrams show the relationships between network components without going into detail about their physical placement. The figures throughout this book are good examples of the sort of graphical representation that is needed. In general, the more complex the network, the more necessary a visual mapping.

Select Protocols

The first decision to make when designing a network is whether it will include only one protocol or more than one protocol. As explained in Chapter 2, "SQL*Net Version 2 Architecture," SQL*Net runs above TNS, which in turn runs over a transport level protocol, with an Oracle Protocol Adapter acting as an interface between TNS and the protocol of choice. The specific hardware below the transport layer is irrelevant to how SQL*Net functions.

You may be able to choose a single transport level protocol that works well on all the components in your network. Protocol adapters are available for most of the major protocols on many platforms. If you have only one protocol in your network, as shown in Figure 3 – 1, then all the components are members of the same *community.*

Figure 3 – 1 A Single Community Network

For reasons of necessity or efficiency, you may choose to have more than one protocol running in your network. You may do this by having multiple protocol adapters on the computers in your network, or, more efficiently, you may have an Interchange between the computers running one protocol and the computers running another. Using SQL*Net and the MultiProtocol Interchange, individual computers can communicate across the protocols that are most compatible with their operating systems. For example, you can have personal computers running Novell's SPX/IPX connected to a VAX server that uses the DECnet protocol. If your network uses one or more Interchanges, as shown in Figure 3 – 2, it is a multi–community network.

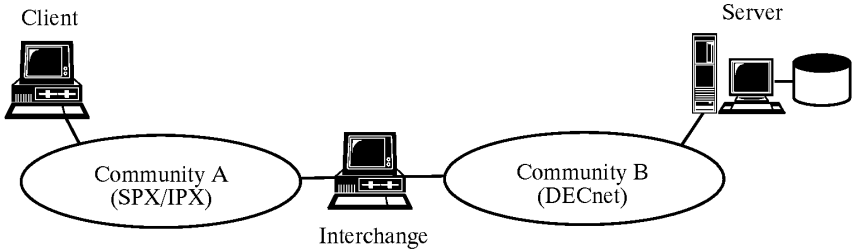

Figure 3 – 2 A Multicommunity Network

Choose Nodes as Interchanges

If you decide on a multi–community network, you must choose what nodes to use for Interchanges to connect the communities. Considerations include:

- What computers work well on all the protocols they connect?
- What computers have the capacity to handle the anticipated traffic?
- Should the computers chosen be dedicated to the Interchange service, or can they handle other applications as well?

Intchg

For further information about Interchanges, see the *Oracle MultiProtocol Interchange Administrator's Guide*.

Choose the Location of the Network Manager

The configuration files for SQL*Net and the other Oracle networking products are created and maintained by using Oracle Network Manager. This product, which is included with SQL*Net release 2.3, provides a graphical user interface through which the network administrator can

quickly and efficiently create and modify configuration information for the network components. The Network Manager requires a workstation running Microsoft Windows. In addition, if your network includes Oracle Names, the Network Manager must have access to an Oracle database.

Select a location for the Network Manager from which it is relatively easy to transfer configuration files to other network components. Network Manager includes a utility, NetFetch, which helps you do this, but a SQL*Net network (version 2 or later) must be up and running before it can be used.

NetMan

For further information about using Oracle Network Manager to create SQL*Net configuration files, see the *Oracle Network Manager Administrator's Guide*

Decide on the Structure of Network Administration

Most networks have one central point of administration, that is, one installation of Oracle Network Manager. However, if you are using Oracle Names and your network is quite large or widely distributed geographically, you may choose to have several points of network administration. For example, if your enterprise–wide network includes both the United States and Europe, you might want to have administrative decisions about the network made locally. A network administrator in Chicago could have jurisdiction over the names and locations of network services in the United States, while someone in Brussels would be responsible for administrative decisions about the network in European countries.

Alternatively, if your network is not very large and dispersed, you may choose to use Oracle Names with the Dynamic Discovery Option. If so, you may not need to use Network Manager at all, because very few configuration files are needed. See the following section for more information about this option.

ONames

For further information about centralized and decentralized administration, see the *Oracle Names Administrator's Guide.*

Choose Whether to Use Oracle Names and the Dynamic Discovery Option

Oracle Names version 2, which is included with this release of SQL*Net, contains an option that provides dynamic registration of servers with well–known Names Servers on the network and automatic replication of data between Names Servers. If you use Oracle Names to provide a

naming service for your network, you must decide whether to use the Dynamic Discovery Option.

If you use Oracle Names, with or without the Dynamic Discovery Option, you must decide what nodes should contain Names Servers, which provide name and address information to enable connections throughout the network. You should have a Name Server in every community. In general, Oracle recommends that Names Servers in a multi–community network be placed on Interchange nodes, thereby minimizing the number of Names Servers required. A Names Server located on an Interchange node can serve both communities.

Oracle Names with the Dynamic Discovery Option

If you choose to use the Dynamic Discovery Option in your network, you may not need to use Oracle Network Manager to create configuration files. If you are using SQL*Net for the first time, and if you are willing to accept all default parameters, with the Dynamic Discovery Option, the only configuration file needed is a LISTENER.ORA for each listener, and that file is created as part of the installation procedure.

If you are adding SQL*Net release 2.3 to an existing network, or if you have a lot of non–default parameters in your configuration files, then you may want to continue to use Oracle Network Manager even if you are using Oracle Names release 2.0 and the Dynamic Discovery Option.

Use Oracle Names release 2.0 and the Dynamic Discovery Option if you are installing SQL*Net for the first time, and if you expect your network to grow and change. If you already have an established network, you can add Oracle Names release 2.0 and the Dynamic Discovery Option if you wish. As your network grows, all new nodes that use SQL*Net 2.3 can use the dynamic registration option while the rest of the network uses configuration files generated by Oracle Network Manager.

ONames

For further information about Oracle Names, see the *Oracle Names Administrator's Guide.*

Naming Considerations

When you name objects such as databases in networked environments, you need to make sure they are unique within the network. This can be a challenge if your company is large, and network administration is not handled centrally. You may be able to guarantee that all services in your building or jurisdiction have unique names, but that does not guarantee that the name is unique within the corporation. Your goal should be to avoid the occurrence of duplicate names if multiple independent TNS communities are joined by installing an Interchange between them.

Domain Names

A recommended network naming technique is to use hierarchical groups or *domain*s in which each administrative unit is assigned to a unique domain based on the function it provides. Many of the examples in this guide feature the fictitious company,ACME Inc., which has segregated its naming domains as shown in Figure 3 – 3.

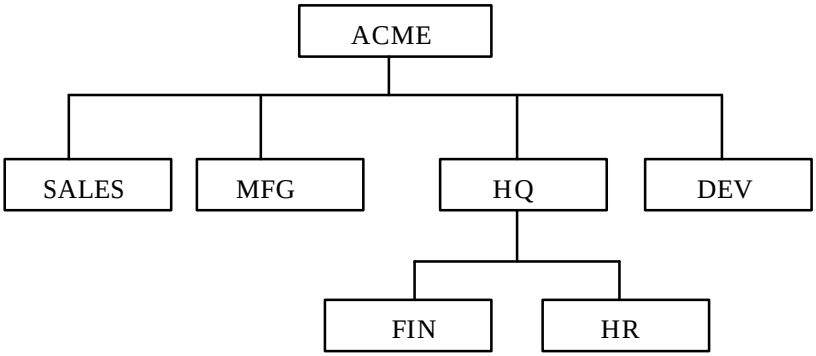

Figure 3 – 3 Naming Domains at ACME

In this figure, each of the boxes represents a separate domain. The domains are related hierarchically; that is, FIN and HR are the children of HQ, which is one of the children of ACME. A network object (such as a database server or an Interchange) within a given domain has a unique name within that domain. This is generally manageable because one or at most a few people have authority to pass out names for that domain. The *global name* for that object includes the parent domains.

For example, consider the corporation shown in Figure 3 – 3. The sales organization (the SALES domain) could have a server named VAULT. The Human Resources group (the HR domain) could also have a server named VAULT. The global names of these servers would be unique. The Sales group's server would have the global database name:

VAULT.SALES.ACME

while the global database name of the Human Resources server would be:

VAULT.HR.HQ.ACME

Names can go to the company level (the ACME stem) or can go to the Internet level (for example, the ACME.COM stem) in support of inter–company communications such as mail or Electronic Data Interchange (EDI). If your organization belongs to the Internet, or you expect that it might join the Internet in the future, the domain names should include the appropriate stem (such as COM, GOV, or EDU).

Note: This naming structure applies whether you are using TNSNAMES.ORA name lookup files, Oracle Names Servers, or Oracle Native Naming Adapters (in conjunction with native name services) to resolve names to addresses. Every network object must have a global object name such as VAULT.HR.HQ.ACME.

If your organization already has global naming conventions, your network components should follow those conventions.

NetMan

Note: Refer to the *Oracle Names Administrator's Guide* for a more detailed discussion of naming conventions.

Default Domain .WORLD Appended to Network Components

Oracle Network Manager automatically appends the domain .WORLD to the name of all network components unless you provide alternative domain names. See the *Oracle Names Administrator's Guide* for further information.

Hierarchical Domains Not Used by the Dynamic Discovery Option

If you are using Oracle Names release 2.0 and the Dynamic Discovery Option, the object names you provide are not interpreted as including hierarchical domains. A name such as VAULT.HR.HQ.ACME or PROD.WORLD is interpreted as a flat name that happens to include dots (.). Therefore, you must be particularly careful in assigning unique names in the network.

Note: If you are using the DDO in a network that has SQLNET.ORA files with default domain parameters included (from an earlier network definition), no default domain will be added to the object name if it contains a dot (.).

Default Domain

Unless you are using Oracle Names and the Dynamic Discovery Option, every client and server has a default domain listed in its SQLNET.ORA file. The default domain is the domain to which the majority of the clients' connection requests are directed. When clients connect to servers in the default domain, they do not need to specify fully qualified service names in their connection requests. For example, a client can connect to a database in its default domain with the service name (global database name) PROFITS.SALES.ACME.COM by specifying the following command:

```
% sqlplus scott/tiger@profits
```

However, if the PROFITS database is not in the client's default domain, the client needs to specify the fully–qualified service name (global database name) in the connection request:

```
% sqlplus scott/tiger@profits.sales.acme.com
```

Note: The default domain is not necessarily the same as the domain of the client itself. For example, clients in one domain may frequently access Oracle servers in another domain.

If you have a network configured for an older version of SQL*Net and you upgrade to a SQL*Net version 2.1 (or later) network, the older names are changed to include a domain. If you don't provide any domain information for the network, when you upgrade to release 2.1 or 2.2, the Oracle Network Manager provides the suffix .WORLD, the default flat domain name, to the network component names.

For example, you might have a database service name defined in version 2.0 as HRFACTS. The same database in release 2.1 and 2.2 would be defined as HRFACTS.WORLD.

Oracle Network Manager automatically includes the following parameter in the SQLNET.ORA file:

```
NAMES.DEFAULT_DOMAIN=WORLD
```

This parameter allows the version 2.0 database service name to be recognized as the same as the database service name defined in the release 2.1 (or later) network.

Note: If you are using Oracle Names release 2.0 and the Dynamic Discovery Option, no SQLNET.ORA file is automatically generated for each client and server, and therefore there is no default domain. If you want to include a default domain for compatibility with other parts of the network, you must create the appropriate parameter in a SQLNET.ORA file. To do so, you can either use Oracle Network Manager to create a SQLNET.ORA file for a group of similar clients, or use the SQLNET.ORA Editor, which is part of the Client Status Monitor, to create SQLNET.ORA files for individual clients. The Client Status Monitor is described in Chapter 4 of the *Oracle Network Products Troubleshooting Guide.*

Diagnostics

ONames

NetMan

See the*Oracle Names Administrator's Guide* and *Oracle Network Manager Administrator's Guide* for further information.

Integrating Oracle into Your Native Naming Environment

Oracle Native Naming Adapters enable network administrators to integrate Oracle service names into their existing naming environment by storing Oracle service names in non–Oracle name services such as Network Information Services (NIS), DCE CDS, Banyan's StreetTalk, and Novell's NetWare Directory Services (NDS). Clients can then access

Oracle network services stored in these naming services. Oracle Native Naming Adapters can be used instead of or in conjunction with Oracle Names Servers and TNSNAMES.ORA files to provide cross–environment name resolution. Service names can be resolved to network addresses by Oracle Names Servers, TNSNAMES.ORA name lookup files, or by a native naming service.

What Is a Name Service?

A name service maps, or ”resolves,” a network name to an address. A name service such as Oracle Names or Network Information Services (NIS) manages and tracks only two attributes—name and location. A client can enter a name and have it resolved to an address, but the client cannot enter an address and have it resolved to a name.

Benefits of Using Name Services

The benefits of using a name service are:

- Name services make administration of larger networks easier and less time–consuming.
  - A name service names each network object, thus enabling a user, administrator, or developer to refer to the network object by name without regard to its physical location.
  - Centralized repositories reduce complexity and mass duplication of information.
  - A names service provides consistent information across the enterprise.
- Network services can be relocated without any change made to clients. Database servers can be relocated without any effect on users.
- Changes can be made in a central location somewhere in the network on a database server instead of having to be stored on each client and server.
  - Changes can be made in a single location and made available to all users.
  - In the case of Oracle Names, whenever a change is made to an existing server or a new server is added to the network, the change is made centrally through Network Manager. This eliminates the need for an administrator to make changes to what potentially could be hundreds or even thousands of clients.
- Name services provide a single unified naming space for all network objects or entities. They uniquely identify all network resources.

- Name services provide a way for people, resources, and applications on the network to find each other without having to know each other's network address—instead people can just specify a simple, user–friendly name.

Oracle Native Naming Adapters Let Users Access Oracle Services Stored in Native Naming Environments

Most large distributed computing systems have several naming services. Maintaining these is a significant task, for which administrators often spend a great deal of time learning and developing tools. Adding another service, such as Oracle Names, requires additional learning, modifying old tools or adding new ones. A typical administrator might ask "Why doesn't Oracle use one of the naming services I already have? That way I can use the tools I already have, and maintain it in ways I'm already familiar with."

Oracle Native Naming Adapters, released with SQL*Net release 2.2 and later, resolve service names stored in customers' native (non–Oracle) naming services. They include:

- OSF DCE CDS (Cell Directory Service)
- Sun's Network Information Service (NIS), formerly known as "Yellow Pages"
- NetWare Directory Service (NDS), Novell's directory service
- Banyan's StreetTalk

Native naming adapters planned for future releases of SQL*Net are:

- Internet Domain Name Service (DNS)
- X.500: The ISO Standard Directory Service originally built for use with OSI

After administrators load Oracle service names and addresses into the name service, users can connect to databases and other Oracle services by specifying the Oracle service name.

Integrating Different Names Services into your Network Environment

Oracle Names can be used in the network along with the native naming service, to provide cross–environment naming integration. Customers who use several different name services in different parts of their environment can achieve connectivity for all Oracle services by using Oracle Names.

Two–Tier Model Allows Distributed Computing

There have been various approaches to using name services in distributed environments. The industry is converging on a common two–tier model for implementing name services, however, with local name services such as Oracle Names, DCE CDS, or NIS handling local domains and communicating with each other through a global name

service such as DNS or X.500. This model consists of an upper–tier global name service and a lower–tier local name service.

Global Name Service

The upper tier is a global name service that provides a standard interface and interoperability between local name services (which may be standard, that is "open," or proprietary). The global name service is based on services such as the Internet's DNS (Domain Name Service) or X.500.

For example, DNS might be used as the global name service, and Oracle Names, TNSNAMES.ORA files, NIS, and DCE CDS might be used as local name services. In this case, the local name service could be open, as is DCE CDS, or it could be proprietary, as is Oracle Names, NIS, and Banyan's StreetTalk. Alternatively, Oracle Names could be used as the global name service if only Oracle services are used, as it works across heterogeneous networks with a mix of protocols such as TCP/IP, DECnet, and SPX/IPX.

Local Name Service

The lower tier can consist of a single local name service or several local name services. For example, Oracle Names could be used as a single local name service, or in combination with other local name services, such as NIS, CDS, Novell's NDS, or Banyan's StreetTalk.

When to Use TNSNAMES.ORA, Oracle Names, or Oracle Native Naming Adapters

If you have a simple distributed network with a small number of services (about 100 or fewer), TNSNAMES.ORA files are a good solution. TNSNAMES.ORA files can resolve service names to network addresses across networks running different protocols. However, keep in mind that when changes need to be made, they must be distributed to every client and server.

If your network is more complicated and has over 100 services, Oracle Names Servers are a good solution. Because people are typically using a mix of protocols such as TCP/IP, SPX/IPX, DECnet, and NetBIOS, Oracle Names Servers are an appropriate naming solution because, like TNSNAMES.ORA files, they can resolve names across a heterogeneous network running several different protocols. Oracle Names Servers are a simpler solution than TNSNAMES.ORA files because they do not have to be located on every client and server (as TNSNAMES.ORA files do), and typically two or three Oracle Names Servers can service an entire region consisting of thousands of clients. Therefore, administration such as adding or relocating services is made much simpler in that changes only need to be made in a few locations.

If you use Oracle Names release 2.0 with the Dynamic Discovery Option, you may not need to create configuration files for your network components, and new components register themselves with the well–known Names Servers so that expanding a network is very easy.

However, integrating this feature into an existing network is tricky. If you already have a large network and it is relatively static, it may not be worth the trouble to use the Dynamic Discovery Option.

One disadvantage of using TNSNAMES.ORA files or Oracle Names for name resolution is that they both store network names and addresses for Oracle services only. If you have non–Oracle services in your network environnment, you need to store them in name services other than those provided by Oracle, such as NIS, DCE CDS, or DNS.

A disadvantage of native naming services such as NIS, DCE CDS, Novell's NDS, and Banyan's StreetTalk is that they are typically specific to an operating system or specific network environment. Oracle Names, on the other hand, is a name service that spans across the entire environment and provides name resolution regardless of the operating system or network environment installed at the customer site. Oracle Names can be used in heterogeneous networks with different protocols.

Oracle Names Can Be Used as a Local or Global Name Service

Oracle Names can be used as a single local or global naming service. It can be used in conjunction with other proprietary or open naming services to provide cross–environment name resolution. For example, Oracle Native Naming Adapters for DCE CDS, NIS, Banyan's StreetTalk or Novell NDS could be installed on all clients and servers in an enterprise network already running Oracle Names to provide name resolution across multiple name services.

Currently, many organizations are using a global naming service such as DNS or X.500, and using one or more local directory or name services such as Oracle Names, Banyan's StreetTalk, or Novell NDS for their proprietary naming systems. What name service(s) you choose to run in your network environment depends on what types of applications and services are running in your environment.

Because Oracle Names is a proprietary name service, that is, it stores and resolves names and addresses for Oracle databases only, you may want to store all your Oracle services in Oracle Names, and use a directory service such as DNS or X.500 as your global naming service. (Oracle Native Naming Adapters for DNS and X.500 are planned for future releases of SQL*Net.) Since DNS is already used on the Internet, and Oracle Names is patterned after DNS, this might be a likely choice. Most corporations and organizations are already connected to the Internet, and since DNS is the name service used on the Internet, this is a likely option for many organizations. It is certainly not a requirement that you use DNS as your global name service—it just happens to be widely used already.

Administrators Can Load Oracle Service Names into their Native Naming Service

Oracle Native Naming Adapters allow administrators to load Oracle service names into their native name service using tools and utilities with which they are already familiar. For example, after installing the DCE CDS naming adapter on all clients and servers in a network environment, the administrator can use familiar DCE utilities to load Oracle service names into the DCE environment. After this is done, users and applications can connect to Oracle applications and services by specifying the Oracle service name or alias.

Oracle

OS Doc

After configuring Oracle databases with Oracle Network Manager, you generate configuration files, including a NATIVE.ORA file. You then load the NATIVE.ORA file, which contains the service names and addresses, into the native naming service. For example, if you are using NIS, you load the Oracle service names and addresses into a NIS zone map using a TNS2NIS utility provided. For information on how to load service names into the native naming service see the Oracle documentation for your platform.

Connecting to Databases with Oracle Service Name

After the Oracle service names and addresses have been loaded into their native naming service, users can connect to databases by specifying the Oracle service name (global database name). For example:

```
sqlplus scott/tiger@sales.us.oracle.com
```

They can specify just the database name if the database is located in the default domain:

```
sqlplus scott/tiger@sales
```

Multiple Name Services Allowed

With this release of SQL*Net, network administrators can specify that one or more native naming services, such as NIS or DCE CDS, in addition to Oracle Names and TNSNAMES.ORA, be contacted to resolve network names to addresses. Using Oracle Network Manager, administrators can list the naming services in any order of priority.

Prepare to Use the Network Manager

Oracle Network Manager is a product included with SQL*Net version 2.1 and later that enables you to create configuration files without having to type in the precise syntax of the files manually. Oracle Network Manager generates the files based upon standard defaults and the unique information you provide through its graphical user interface. Unless you are using Oracle Names and the Dynamic Discovery Option, you must use Network Manager to create the configuration files.

To use the Network Manager effectively you must have detailed information about the network at hand. This section describes the information you must have ready.

> **Note:** You must use Network Manager to create all SQL*Net configuration files (except PROTOCOL.ORA, which must be created by hand). Configuration files created manually are not supported by Oracle Corporation. You can edit most values in the SQLNET.ORA file using the SQLNET.ORA Editor.

Know the Network

The Network Manager knows the syntax of the configuration files and it knows the default values for parameters in those files. However, it knows nothing about your network until you supply the information. In fact, supplying accurate information to the tool is your main task in using it.

Names

Choose names for the following:

- all domains (unless you are using only one domain—WORLD)
- all communities
- all network listeners, and the computers (nodes) on which they run
- all databases, to act as *database service names*; service names are short identifiers for their connect descriptors

  > **Note:** As of SQL*Net release 2.1 and the Oracle Server release 7.1, the service names you provide must match precisely the unique global database names assigned by the database administrator. To achieve this, it may be necessary to change some of the service names you have been using. For example, if your previously defined SQL*Net release 2.0 network has service names that do not match the global database names, those service names must be changed. Similarly, if the network includes some databases that were named before you established your current domain names, their global database names and service names must reflect the current domain structure.

- all Interchanges, if any, and the computers (nodes) on which they run
- all Names Servers, if any, and the computers (nodes) on which they run

- all client profiles (A *client profile* or *client type* is a group of clients with the same communication requirements.)

  Note: Network Manager automatically generates names for Interchanges, Names Servers, and client profiles. You do not need to supply names for them unless you want to change the Network Manager–generated defaults.

Note that even if you have a one–protocol network, that is to say, a one community network, you must supply a name for that community. The community names should follow global naming conventions. See the section ”Naming Considerations” earlier in this chapter. The name of each component within the network must be unique.

Addresses

Define addresses for the following:

- all servers
- all Interchanges, if any
- all Names Servers, if any

The addresses for these components consist of the names of the communities of which they are a part and protocol–specific information.

Protocol Specific Information

Different protocols require different protocol–specific information. The following table summarizes the keywords for the protocols currently supported in a TNS network.

Protocol	**Keywords**
AppleTalk	SERVICE
ASYNC	PHONENUMBER ASYNC_SERVER LOCAL_LOOKUP
Banyan	ITEM GROUP ORGANIZATION
DECNet	NODE OBJECT
DCE	SERVICE SERVER_PRINCIPAL CELL_NAME (optional)
LU6.2 / APPC	LU_NAME LOCAL_LOOKUP TP_NAME

Named Pipes	PIPE SERVER
NetBIOS	NTBNAME
OSI	NSAP TSEL
or	
	HOST SERVICE
SPX/IPX	SERVICE
TCP/IP	HOST PORT

OS Doc

The Network Manager provides default values for many of these protocol–specific keywords. See the Oracle operating system–specific documentation for your platforms for information on what values to supply for the protocol keywords.

Connect Data

You must also provide the system identifiers (SIDs) for database servers. SID names typically match the database name; however, this is not a requirement.

SQL*Net V1 Connect Strings

If your network includes both SQL*Net version 1 and version 2, have available the names of the files that hold the Version 1 connect strings and their aliases, and know where in the file system they are stored.

Parameters

The LISTENER.ORA file includes a number of required and optional parameters that describe the listener. You should gather the parameter information and have it ready to use in the Network Manager. The parameters for LISTENER.ORA are described in Appendix A, "Contents of the Configuration Files."

MultiProtocol Interchange Information

Intchg

The Network Manager creates configuration files for the Interchanges in your network, if you have them, based on information you supply. The files are described in Appendix A of the *MultiProtocol Interchange Administrator's Guide*.

Names Servers

ONames

If you are using Oracle Names without the Dynamic Discovery Option, the Network Manager creates a NAMES.ORA configuration file for each of the Names Servers in your network. See the *Oracle Names Administrator's Guide* for information about this file.

Multiple Network Managers

If your network includes Oracle Names, you may want to provide further information about the naming structure of your network. You may want to include delegated administrative regions, so that widely

ONames

separated parts of your network have some autonomy in their administration. See the *Oracle Names Administrator's Guide* for further information.

Database Links

If you are using Oracle Names, you may want to specify additional information such as username and password for the global database link that is automatically created for each database defined in Network Manager. See Chapter 5, "Using SQL*Net," in this manual, and Chapter 5, "Entering Component Information" in the *Oracle Network Manager Administrator's Guide*, and *Oracle7 Server Distributed Systems, Volume I* and the *Oracle Names Administrator's Guide* for information on global database links.

NetMan

ONames

Use the Oracle Network Manager

Once you have done the planning, made your decisions, and gathered the information you need, you are ready to configure the network. To do this, use Oracle Network Manager. This tool makes creating the configuration files relatively easy. Equally important, it helps you make them error free.

NetMan

You will find complete instructions about the use of Network Manager in the *Oracle Network Manager Administrator's Guide.*

After you have created and distributed the configuration files, you can start and use the network. Information about using the network is in Chapter 5, "Using SQL*Net".

CHAPTER

4

Migrating from an Earlier Version of SQL*Net

This chapter describes issues the network administrator must resolve when migrating from an earlier version or release of the SQL*Net product. The chapter discusses three issues:

- coexistence of SQL*Net version 1 and version 2.x
- using the Oracle Network Manager to ease the transition to release 2.1 or later
- migrating from Version 2.0 to release 2.1 or later

Coexistence of SQL*Net Version 1 and Version 2.x

> **Note:** SQL*Net version 1 cannot communicate with SQL*Net version 2, but it can coexist with it. Different releases of SQL*Net version 2 can communicate with each other. Therefore, the migration issues from version 1 to version 2, 2.1 or 2.2 are somewhat different from those from 2.0 to 2.1 or 2.2. In this section SQL*Net version 1 is contrasted with SQL*Net version 2.0, 2.1, or 2.2. In fact ,version 2 and later releases are similar in regard to migration issues from version 1.

If you already have a SQL*Net version 1 network, you must decide which of your nodes you will upgrade immediately to version 2.x, and which will continue to use only version 1.

Ideally, all platforms would run SQL*Net version 2.x supporting all protocols, and this section of the manual would not exist. In reality, there will be a period of time during which some platforms will have SQL*Net version 2.x available, and others will have SQL*Net version 1. During the transition, you may want to have some platforms with both SQL*Net version1 and SQL*Net version 2.x installed, so that they can communicate with all other platforms.

> **Note:** SQL*Net version 1 nodes can communicate only with other SQL*Net version 1 nodes; however, version 2 and release 2.1 and 2.2 nodes can communicate with each other.

Version 1 Coexistence

In any SQL*Net installation, for each node there are three options directly related to the type of clients or servers with which each node communicates:

- Only SQL*Net version 1 is installed. This installation assumes:
  - for a client, this node will connect only to servers running SQL*Net version 1
  - for a server, this node will receive connections only from clients running SQL*Net version 1
- Only SQL*Net version 2.x is installed. This assumes:
  - for a client, this node will connect only to servers running SQL*Net version 2.x
  - for a server, this node will receive connections only from clients running SQL*Net version 2.x
- Both SQL*Net version 2.x and SQL*Net version 1 are installed. (You do not need to reinstall SQL*Net version 1 if it is already installed.) This assumes:

– for a client, this node will connect to multiple servers, some running SQL*Net version 1 and others running SQL*Net version 2.x. (This ability is not available on MS–DOS machines.)

– for a server, this node will receive connections from clients running SQL*Net version 1 and SQL*Net version 2.x.

This model allows version 1 service to continue in your existing network until such time as all nodes are running version 2.x.

Distinguishing Version 1 and Version 2.x Connections

The version of SQL*Net that is used for a given connection is determined by the format of the connect string. Where a version 1 connect string is sent, the version 1 software and version 1 listener (on the server) are used. Where a TNS connect descriptor is sent, the version 2.x software and listener (on the server) are used.

Most of the time people use aliases for connect strings (in SQL*Net version 1) or service names for connect descriptors (in SQL*Net version 2.x). The way these aliases and service names are handled is operating system specific. On UNIX and VMS systems, if version 2.x is installed, the version 1 configuration files for mapping aliases to connect strings are not read. Therefore, to continue to use version 1 aliases, you must import them into the version 2.x network definition and generate the TNSNAMES.ORA configuration file. Network Manager makes this easy to do. See Chapter 5 in the *Oracle Network Manager Administrator's Guide* for further information. In general, personal computers will continue to read the version 1 configuration files to connect to the database indicated by the version 1 alias. See the Oracle operating system–specific documentation for your platform for details.

NetMan

OS Doc

Note: If you are using Oracle Names, the TNSNAMES.ORA file is not necessary. Instead, the Names Servers read the destination addresses from the network definition database created by the Network Manager.

Types of SQL*Net Installations

With respect to compatibility with SQL*Net version 1, there are four significant types of SQL*Net version 2 networks:

- a single community with only SQL*Net version 2.x nodes
- a single community with both SQL*Net version 1 and SQL*Net version 2.x nodes

- two or more interconnected communities with only SQL*Net version 2.x nodes
- two or more interconnected communities with both SQL*Net version 1 and SQL*Net version 2.x nodes

Each of these is discussed in the following sections.

Single Community with Version 2.x Nodes

The simplest possible type of SQL*Net version 2.x installation is a single TNS community where all clients and servers use SQL*Net version 2.x for communication. Figure 4 – 1 shows a TNS community based on Novell's SPX/IPX where all of the clients and servers are using SQL*Net version 2.x for SPX/IPX.

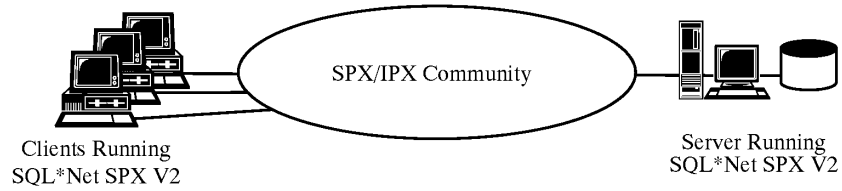

Figure 4 – 1 Community with Only V2.x Nodes

In this example, because SQL*Net version 2.x is installed on both the clients and servers, any client can talk to any server in the community. For all connections, the version 2 syntax on the client and the listener on the server is used.

If SQL*Net version 2.x is available for all of the platforms in your client–server network, and all nodes run the same protocol, this is the layout you would choose.

Single Community with Version 1 and Version 2.x Nodes

During the initial release of SQL*Net version 2.x, there will be platforms on which SQL*Net version 2.x is available, and platforms on which SQL*Net version 1 is still the latest release. This is simply because the porting process on some platforms takes longer than on others.

As a result you may have a network in which both SQL*Net version 1 and SQL*Net version 2.x coexist. Figure 4 – 2 shows a network of TCP/IP hosts, some of which have SQL*Net version 2.x available, and some of which still run only SQL*Net version 1.

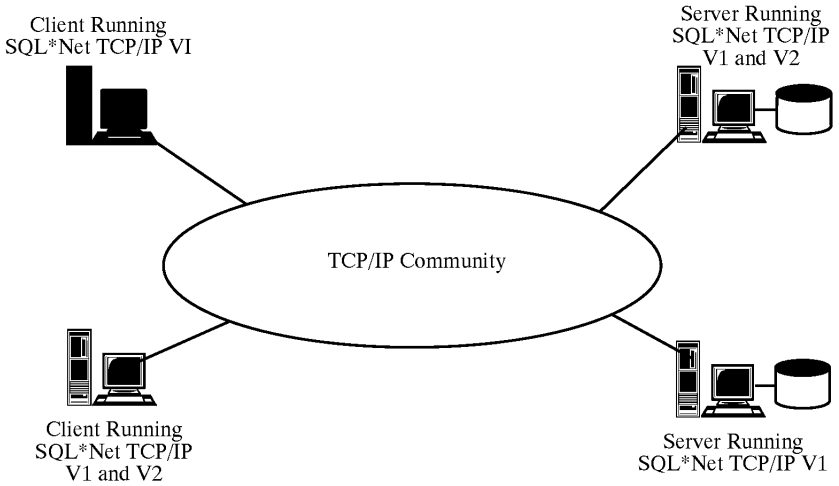

Figure 4 – 2 Single Community with V1 and V2.x Nodes

Each of the nodes in this community may require connectivity with any of the other nodes. Therefore, the nodes running SQL*Net version 2.x also have SQL*Net version 1 installed so that they can communicate with the SQL*Net version 1 nodes.

On nodes that can run only one SQL*Net driver at a time, such as MS–DOS, the lowest common denominator theory applies. If that node connects only to version 2.x servers, it should use SQL*Net version 2.x. If there are still SQL*Net version 1 servers it must communicate with, it should use SQL*Net version 1. However, it will be unable to connect to servers that have only version 2.x running.

Multiple Communities with Version 2.x Nodes

The third noteworthy type of network is a multi–community network running only SQL*Net version 2.x. This is a TNS network of at least two communities in which all of the members use SQL*Net version 2.x. Figure 4 – 3 shows a sample network of clients and servers based on the SPX / IPX and TCP / IP protocols.

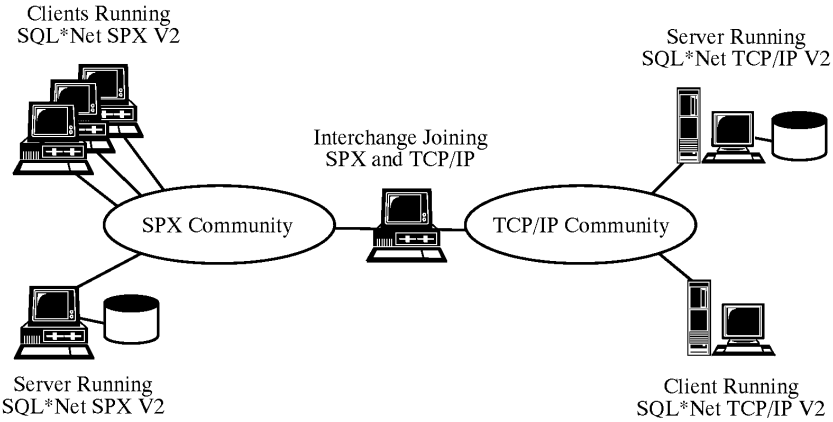

Figure 4 – 3 Multiple Communities with V2.x Nodes

In the network shown, any of the clients can communicate with any of the servers:

- The SPX / IPX clients can connect to the SPX / IPX server. In this case, the Interchange is not used; this connection is similar to the connections in Figure 4 – 1.
- The TCP / IP clients can connect to the TCP / IP server. Once again, the Interchange is not used; this is similar to the connections in Figure 4 – 1.
- The SPX / IPX clients can connect to the TCP / IP server. These connections go from the client over SPX / IPX to the Interchange and from the Interchange over TCP / IP to the server.
- The TCP / IP clients can connect to the SPX / IPX server. These connections go from the client over TCP / IP to the Interchange and from the Interchange over SPX / IPX to the server.

Intchg

For more information on the MultiProtocol Interchange, see the *Oracle MultiProtocol Interchange Administrator's Guide*.

Multiple Communities with Version 1 and Version 2.x Nodes

The last type of network to consider is a network with multiple communities in which not all of the nodes have SQL*Net version 2.x.

Figure 4 – 4 shows a sample network in which a TCP / IP community and an SPX / IPX community are joined by an Interchange.

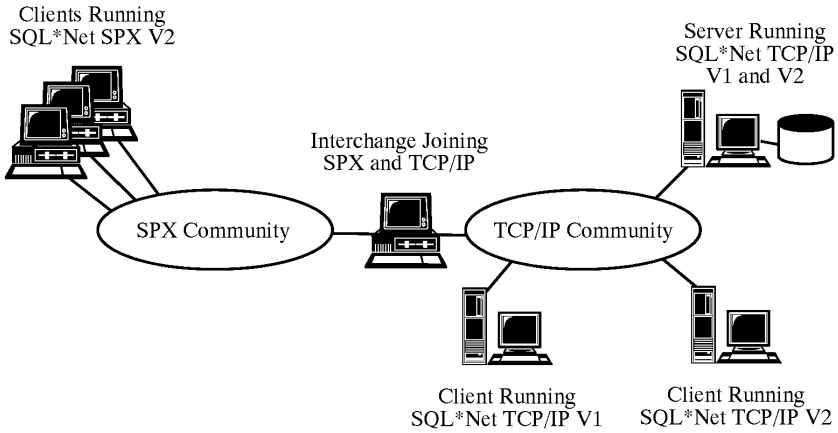

Figure 4 – 4 Multiple Communities with V1 and V2.x Nodes

In the sample network shown, all of the nodes in the SPX/IPX community use SQL*Net version 2.x, but some of the nodes in the TCP/IP community only have SQL*Net version 1 available. All version 2.x servers have version 1 installed as well. In the picture shown, the nodes can communicate as follows:

- The SPX/IPX nodes communicate through SQL*Net version 2.x and the Interchange to the TCP/IP server.
- The TCP/IP nodes communicate among themselves using SQL*Net version 2.x where available; where it is not, they use SQL*Net version 1.

  Note: Using the Interchange for cross–protocol connectivity requires SQL*Net version 2.x on both the client and the server. For example, if there were a server in the TCP/IP community that ran only SQL*Net version 1, the SPX/IPX clients could not access it through the Interchange. Similarly, if there were a server running SQL*Net version 2.x in the SPX/IPX community, the SQL*Net version 1 clients in the TCP/IP community could not access it.

The ability of SQL*Net version 2.x to connect clients and servers in different networks using the MultiProtocol Interchange is a feature unavailable on SQL*Net version 1. However, even in a single-protocol network, where both SQL*Net version 1 and version 2.x are available, there are many good reasons to move toward SQL*Net version 2.x. These reasons include:

- **Better diagnostics**. The logging and tracing features of SQL*Net version 2.x make SQL*Net errors easier to diagnose. The

diagnostics may allow immediate resolution of any problems, and provide additional information if you need the help of Oracle Worldwide Customer Support in resolving a problem.

- **Increased manageability.** SQL*Net version 2.x connect descriptor syntax is consistent over all protocols and easily understood by anyone familiar with the syntax. Also, the method of assigning a service name to a connect descriptor is the same on all platforms. Use Oracle Network Manager to generate configuration files. If you define the databases in the network once through Network Manager, any platform can connect to the databases so identified.
- **Long-term support.** SQL*Net version 1 will not be supported after June 30, 1996, so it is recommended that you complete migration to SQL*Net version 2 before this date. SQL*Net version 1 is no longer being shipped with Oracle7.
- **Support for database features.** SQL*Net version 2 is a requirement for some of the new features of the Oracle7 Server, such as the multi–threaded server. To fully use the features of Oracle7, you must migrate to the latest Oracle7 release, and to fully use the features of SQL*Net you must migrate to the latest SQL*Net release.
- **Future enhancements.** Future Oracle network products will be designed to enhance the functionality of SQL*Net version 2.x only. For example, Secure Network Services works only with SQL*Net release 2.1 or later.

Upgrading from Version 1 to Version 2.x

The act of upgrading from SQL*Net version 1 to version 2.x requires few actual changes. You mustdo the following:

- Replace each version 1 connect string with a version 2 connect descriptor. Use the Network Manager to create the connect descriptor in the appropriate configuration file, TNSNAMES.ORA, to store them in the network definition on a database for the use of Oracle Names, or to store them in a native naming service such as NIS or DCE's CDS.
- Update embedded database links on the servers.
- On a multi–user system, relink any 3GL programs that you wish to use with SQL*Net version 2.

Locating Connect Strings

Before you can update SQL*Net version 1 connect strings into SQL*Net version 2.x connect descriptors you must first understand the scope of the potential change. Depending on your SQL*Net version 1 installation, there may be connect strings in both the client application and server, and particularly on the client, they can be in any number of places.

Application Connect Strings

For example, at the application, connect strings can be found in:

- the SQL*Net version 1 alias file for that platform (for an Oracle RDBMS version 6 or earlier.)

  Most platforms have a means of assigning a single name to represent a connect string to simplify its use. See your Oracle operating system–specific documentation for the SQL*Net version 1 drivers on your platform.

OS Doc

- operating system-specific program startup scripts

  Commonly an operating system startup program can start another program to simplify a complex command string. For example, an operating system startup program called THINK might start an application that connects to a database with SQL*Forms using a command such as:

  ```
  runform think scott/tiger@connect_string
  ```

- a menu-based program such as SQL*Menu, or a similar product on a given operating system

  These programs allow an operating system command like the one in the THINK example above to be presented as a menu option.

- embedded in applications

  Some non–Oracle products allow access to Oracle data by way of connect strings embedded in the application. For example, DDE Manager allows Microsoft Excel to access Oracle data from a spreadsheet; to do this, the spreadsheet may contain a connect string.

Server Connect Strings

On the server, connect strings are usually found only in the data dictionary tables USER_DBLINKS or DBA_DBLINKS, where they are inserted by a CREATE DATABASE LINK statement.

Changing Connect Strings

This section lists the procedure for migrating your SQL*Net version 1 connect strings to SQL*Net version 2.x connect descriptors:

1. Consider the requirements for your network, identifying which nodes will be running SQL*Net version 1, version 2.x, or both.

2. Determine where and how SQL*Net version 1 connect strings are used in your installation. Find how they are used at the system level and at the user level. (For example, connect strings may be stored in a global TNSNAMES.ORA accessible by everyone in the network, as well as in individual users' private TNSNAMES.ORA files.)
3. Install SQL*Net version 2.x following the instructions in the Oracle documentation for your platform.
4. Create the necessary configuration files using Oracle Network Manager. If you are adding SQL*Net version 2.x support to an existing SQL*Net version 1 installation, SQL*Net version 1 operation will be unaffected; that is, the version 1 listener and configuration files will function as before. The TNSNAMES.ORA configuration file (or Oracle Names or a Native Naming service such as NIS) contains service names that act as aliases for the SQL*Net version 2.x connect descriptors for each database.

   Note: Service names and global database names are often the same; for example, HR.US.ACME.COM.
5. Distribute the TNSNAMES.ORA file so that all clients and servers running SQL*Net version 2.x have access to it.

   Note: If the network includes Oracle Names, the service names and connect descriptors are stored in a database and resolved by Oracle Names, so TNSNAMES.ORA is not necessary. Similarly, if Oracle service names are stored in a native naming service such as NIS or DCE's CDS, then TNSNAMES.ORA files are not necessary.
6. On all version 2.x clients, change all version 1 connect strings to the equivalent service names mapped to version 2.x connect descriptors if the node the clients reference can run version 2.x. Not all nodes may be upgraded, but the ones that are should use SQL*Net version 2.x.
7. On all version 2.x servers, change all database links using version 1 to database links using version 2.x when they reference another server that can run version 2.x. This usually means dropping and re-creating the database link in the server. The new database link should use the service name that has been defined in the TNSNAMES.ORA file (or entered into Network Manager for use by Oracle Names or by a native naming service such as NIS). Any two servers that run version 2.x should use version 2.x to communicate.

NetMan

Note: If the network includes Oracle Names, global database links are available, and can be edited using the Network

Manager. See Chapter 5 in the *Oracle Network Manager Administrator's Guide* for details.

8. Test the new configuration. Make sure the new reference works as it did with SQL*Net version 1.

Sample Upgrade for Two Nodes

This section provides a simple example of the eight-step procedure described in the previous section, "Changing Connect Strings". If you think in terms of pairings, upgrading any number of nodes from SQL*Net version 1 to version 2.x should not be difficult.

This example uses three nodes, one client and two servers, as shown in Figure 4 – 5. It shows how the client changes to access either server over version 2.x, and how the DETROIT server changes to access the CLEVELAND server.

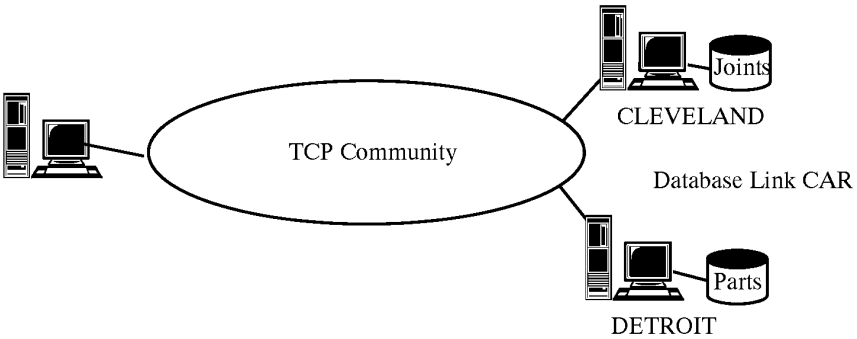

Figure 4 – 5 Sample Nodes for V2 Upgrade

1. First, determine that all three nodes will be upgraded to SQL*Net version 2.x.

   The client node uses SQL*Net version 1 connect strings in operating system-specific startup files and within SQL*Menu. They currently appear as follows:

   For the CLEVELAND server:

   ```
   T:CLEVELAND:JOINTS
   ```

   For the DETROIT server:

   ```
   T:DETROIT:PARTS
   ```

   In addition, the CLEVELAND server has a database link connecting it to a table in the DETROIT server as follows:

```
CREATE DATABASE LINK CAR
CONNECT TO MOTOR IDENTIFIED BY CITY
USING 'T:DETROIT:PARTS'
```

2. Perform the software installation as described in the Oracle operating system-specific manual for the platforms involved.

NetMan

3. Use Oracle Network Manager (described in the *Oracle Network Manager Administrator's Guide*) to create the configuration files, including LISTENER.ORA, TNSNAMES.ORA, and SQLNET.ORA. In this example, the resulting TNSNAMES.ORA file would have the following entries, where CLEVELAND and DETROIT are service names mapped to their respective connect descriptors:

```
CLEVELAND.SALES.ACME =
  (DESCRIPTION=
       (ADDRESS=
       (      COMMUNITY=TCP.SALES.ACME)
              (PROTOCOL=TCP)
              (HOST=CLEVELAND.SALES.ACME)
       (      PORT=1521))
       (CONNECT_DATA=
              (SID=JOINTS)))

DETROIT.SALES.ACME =
  (DESCRIPTION=
       (ADDRESS=
              (COMMUNITY=TCP.SALES.ACME)
              (PROTOCOL=TCP)
              (HOST=DETROIT.SALES.ACME)
              (PORT=1521))
       (CONNECT_DATA=
              (SID=PARTS)))
```

4. Distribute the configuration files so that they are accessible to the appropriate clients and servers, as described in the *Oracle Network Manager Administrator's Guide.* Note that all nodes refer to a given server the same way. See the Oracle operating system-specific documentation for your platforms for the correct locations of the files.

OS Doc

Note: If you are using Oracle Names, the TNSNAMES.ORA file is unnecessary. The service names and connect descriptors are stored in a database and accessed by Oracle Names. Similarly, if you have configured an Oracle Native Naming Adapter for a client type, service names are stored in a native naming service such as NIS.

Refer to the *Oracle Names Administrator's Guide* and the *Oracle Network Manager Administrator's Guide* for further information.

5. Change the client startup file and SQL*Menu entries to refer to the version 2 service names:

```
runform max/python@DETROIT.SALES.ACME
runform max/python@CLEVELAND.SALES.ACME
```

6. Change the database link definition in the CLEVELAND server :

```
CREATE DATABASE LINK DETROIT
CONNECT TO MOTOR IDENTIFIED BY CITY
USING 'DETROIT'
```

Because in SQL*Net version 2 the database link name must match the global database name, the database link name must be changed from CAR to DETROIT. In this example, the CLEVELAND server and the DETROIT server are in the same domain; therefore you do not need to include the domain in the CREATE DATABASE LINK statement.

ONames

NetMan

Note: If you are using Oracle Names, global database links are automatically created between all database servers. You can edit global database links through the Network Manager. Refer to the *Oracle Names Administrator's Guide* and the *Oracle Network Manager Administrator's Guide* for further information. You can add a username and password (CONNECT TO data) and connection qualifiers to global database links by using the Network Manager. See the section "Database Links with Oracle Names" in Chapter 5 of this manual.

With Oracle Names, you could create a database link named CAR. It would require two steps:

- First, use the Network Manager to create the alias CAR for the service name DETROIT.SALES.ACME.
- Then create a database link from CLEVELAND to DETROIT using CAR as the link name. You do not have to include a USING clause because Oracle Names finds the service name that CAR is mapped to in the network definition database.

```
CREATE PUBLIC DATABASE LINK CAR
CONNECT TO MOTOR IDENTIFIED BY CITY
```

7. At the client, run an application that uses the database link DETROIT. If the client connects to the CLEVELAND server and the database link returns the data, then those components are

functioning correctly. Run a similar test of the client / server link between the client and the DETROIT server.

Coexistence of Version 1 and Version 2

If you have a large number of SQL*Net version 1 connect strings to update, you can install SQL*Net version 2.x beside SQL*Net version 1 in both the clients and servers as a temporary method of allowing version 2.x functionality to coexist with version 1 connect strings. All existing connect strings will continue to function within their version 1 limitations. This may make the migration more manageable.

> **Note:** It is not possible to have both SQL*Net version 1 and SQL*Net version 2 running simultaneously on an MS–DOS machine.

Relinking 3GL Applications

OS Doc

All user–written 3GL programs that perform connections to a remote database must be relinked with the SQL*Net version 2.x libraries appropriate for your platform. For details refer to your Oracle platform specific installation guide for SQL*Net version 2.x.

Moving from SQL*Net Release 2.0 to a Later Release

SQL*Net release 2.3 is designed to be backward compatible with previous SQL*Net 2.x releases. The network may include nodes running SQL*Net release 2.0, 2.1 or 2.2. If you have a SQL*Net release 2.0 network up and running at your site, the configuration files you have distributed will continue to be valid after release 2.3 is installed. However, you will not be able to take advantage of some of the new features of release 2.3 without recreating the files using Oracle Network Manager release 3.1, which is included with SQL*Net release 2.3. Specifically, you must create new files to use listener load balancing, and to use Network Manager in conjunction with Oracle Names release 2.0 and the Dynamic Discovery option.

OS Doc

SNS

SNMP

For information on configuring Oracle Native Naming Adapters see your operating system–specific documentation. See the *Secure Network Services Administrator's Guide* for information on configuring Secure Network Services. See Chapter 6, "Configuring Oracle SNMP Support" in this guide for information on configuring the SNMP Support feature and the *Oracle SNMP Support Reference Guide.* For information on SQL*Net / DCE, see the *SQL*Net/DCE Administrator's Guide.*


NetMan

The Network Manager includes a utility, NETCONV, to make the migration from version 2.0 to release 2.1 or later easy. See Appendix C in the *Oracle Network Manager Administrator's Guide* for further information. If you are upgrading your network from release 2.1 or 2.2 to 2.3, you will not need to use NETCONV—Network Manager will convert the network definition automatically for you.

Attention: When you use Oracle Network Manager, the default is to encrypt the passwords in LISTENER.ORA. If your network includes some nodes that are running SQL*Net release 2.0, be careful that the passwords for those listeners are not encrypted. Encrypted passwords are not recognized by SQL*Net release 2.0.


ONames

Note: For information about using Oracle Names release 2.0 with the Dynamic Discovery option in a network that includes various versions of SQL*Net, see Chapter 5 in the *Oracle Names Administrator's Guide.*

CHAPTER

5 Using SQL*Net

This chapter describes how to use SQL*Net release 2.3 after it has been configured. It includes:

- starting and testing the network configuration
- common errors during testing
- using the Listener Control Utility
- initiating a SQL*Net connection
- distributed database management
  - database links
  - synonyms
  - snapshots
  - copying data between databases
- operating system-authorized logins

Testing the Network Configuration

> **Note:** This section assumes that you are not using Oracle Names 2.0 with the Dynamic Discovery Option enabled. If you are using the Dynamic Discovery Option, please refer to the *Oracle Names Administrator's Guide* for information about starting the network.

Configure the network using the Oracle Network Manager. Once the configuration files are on the destination machines, each component can be started and tested. The preferred sequence for testing the network is to:

- start and test each Oracle Names Server (if included)
- start and test each listener
- start and test each Interchange (if included)
- test each client type

Start a Names Server

The STARTUP command of the NAMESCTL utility loads the Names Server into memory and tells it to begin executing. At startup, the Names Server loads its configuration, loads its data, then becomes available to answer requests.

```
NAMESCTL> STARTUP
```

Test a Names Server

To test that a Names Server is operating correctly, use the PING command. Following are two ways to PING the server LABRADOR in the US.ACME domain. Alternatively, you can use TNSPING (described on page 5–5).

```
NAMESCTL> PING LABRADOR.US.ACME
or
NAMESCTL> SET SERVER LABRADOR.US.ACME
NAMESCTL> PING
```

You can also PING multiple Names Servers by using a single command, for example:

```
NAMESCTL>PING HUEY.UK.ACME DUEY.UK.ACME LOUIE.UK.ACME
```

PING responds with the time it takes to contact the Names Server and return an acknowledgment.

If PING fails, make sure the Names Server has been started. If it has been, double-check the configuration in Network Manager, especially the defined address of the Names Server in question.

Start the Listener

From each listener's node, use the Listener Control Utility, LSNRCTL, to start each listener. In command line mode, to start a default listener (which is a listener defined in a LISTENER.ORA file with the alias LISTENER):

```
LSNRCTL START
```

To start a non–default listener:

```
LSNRCTL START listener_name
```

LSNRCTL will display a status message indicating that it has started successfully. After LSNRCTL has started the listener, it has no further control over the listener. Check that all expected SIDs for that listener are listed in the services summary in the status message. For more information on the Listener Control Utility see the section "Using the Listener Control Utility" later in this chapter.

Test the Listener

To test the listener, initiate a connection from a client in the same community as the listener to any active database controlled by that listener.

The simplest test uses SQL*Plus as follows:

```
SQLPLUS user/password@service_name
```

The *service_name* may be found in the TNSNAMES.ORA file, an Oracle Names Server, or a native naming service such as NIS or DCE's CDS. For more information on testing SQL*Net from a client, see the section "Initiating a SQL*Net Connection" later in this chapter.

If there are no clients in the same community as the listener, you must start an Interchange before testing the listener.

Repeat these steps for each listener in the network.

Start theInterchange

Use the Interchange Control Utility, INTCTL, on the Interchange node to start an Interchange. (You cannot run INTCTL from a remote node.) For example, in command linemode the command is:

```
INTCTL START INTERCHANGE
```

or

```
INTCTL START INT
```

Intchg

INTCTL displays a status message indicating that it has started successfully. For more information on the Interchange Control Utility see Chapter 6, "Controlling the MultiProtocol Interchange," in the *Oracle MultiProtocol Interchange Administrator's Guide*.

Test the Interchange

To test the Interchange, initiate a connection through the Interchange from a client in one community to a database in another. On the client machine type:

```
SQLPLUS user/password@service_name
```

The *service_name* for the database in the other community may be stored in TNSNAMES.ORA, an Oracle Names Server, or a native naming service such as NIS or DCE's CDS. To ensure that the connection went through the Interchange, type:

```
INTCTL STATUS INTERCHANGE
```

The Connection Manager should indicate that there is one active connection through the Interchange. You can run the STATUS command on the Interchange machine. If the Interchange is listed in the TNSNAMES.ORA file, stored in an Oracle Names Server, or stored in a native naming service, then you can run the STATUS command remotely from any other node.

Test Network Objects Using NAMESCTL

To test that a network object exists, use the QUERY comand. The syntax for this command is as follows:

```
NAMESCTL> QUERY global_object_name type
```

Database service names have the type A.SMD, and database links have the type DL.RDBMS.OMD. The following example shows a query of the database service name BUGSY in the MACS.ACME domain.

```
NAMESCTL> QUERY BUGSY.MACS.ACME A.SMD
```

The QUERY command returns the amount of time the transaction took and information about the network object.

Test Network Objects Using TNSPING

The network administrator can use TNSPING to determine whether or not a service on a SQL*Net network can be successfully reached. The service being contacted can be an Oracle database, an Oracle Names Server, or any other Oracle (TNS) service.

If you can connect successfully to a TNS service using TNSPING, it displays an estimate of the round trip time (in milliseconds) it takes to reach the Oracle service. If it fails, it displays a message describing the error that occurred. This allows you to see the network error that is occurring without the overhead of a database connection.

You invoke TNSPING on the command line as follows:

```
tnsping service_name count
```

On some platforms, such as Microsoft Windows, the interface may be different, but the program accepts the same arguments.

- *service name* must exist in TNSNAMES.ORA, Oracle Names, or the name service in use, such as NIS or DCE's CDS. (For service names to be resolved to addresses in a native naming service such as NIS or DCE's CDS, Oracle Native Naming Adapters must be installed on clients and servers. See your Oracle operating system–specific documentation for information on installing and configuring Oracle Native Naming Adapters.)
- *count* (optional) determines how many times the program attempts to reach the server.

If the service name specified is a database name, TNSPING attempts to contact the corresponding network listener. It does not actually determine whether or not the database itself is running. (To do this, use Server Manager to attempt a connection to the database.)

Examples of the Use of TNSPING

Following are some examples of how a network administrator or user can use TNSPING. For example, to connect to a database named SPOTDB, the command:

```
tnsping spotdb
```

produces the following message:

```
TNS Ping Utility for SunOS:  Version 2.2.2.0.0 – Production on
10–Mar–95 10:09:13
Copyright (c) Oracle Corporation 1995.  All rights reserved.
Attempting to contact
(ADDRESS=(COMMUNITY=build_tcp.us.oracle.com)(PROTOCOL=TCP)
(HOST=spot)(PORT=1599))
OK (50msec)
```

To check whether an Oracle Names Server can be reached, use a command similar to the following:

```
tnsping (ADDRESS=(COMMUNITY=build_tcp.us.oracle.com)
(PROTOCOL=TCP)(HOST=fido)(PORT=1600))
```

The address of the Oracle Names Server is located in SQLNET.ORA (in the NAMES.PREFERRED_SERVERS and NAME.PREFERRED_SERVERS parameters) on the client.

> **Note:** Both parameters showing the client's preferred Names Server must be included for backward compatibility with an earlier version of the product.

A message similar to the following will be returned to the user:

```
TNS Ping Utility for SunOS: Version 2.2.2.0.0 – Production on
10–Mar–95 10:09:59
Copyright (c) Oracle Corporation 1995. All rights reserved.
```

```
Attempting to contact
(ADDRESS=(COMMUNITY=build_tcp.us.oracle.com)
(PROTOCOL=TCP)(HOST=fido)(PORT=1600))
OK (70 msec)
```

To determine whether the STPRD database can be connected to, and to specify that TNSPING try to connect 10 times and then give up, use the following command:

```
tnsping stprd 10
```

This command produces the following message:

```
TNS Ping Utility for SunOS:  Version 2.2.2.0.0 – Production on
10–Mar–95 10:10:28
Copyright (c) Oracle Corporation 1995. All rights reserved.
Attempting to contact (ADDRESS=(COMMUNITY=build_tcp.us.oracle.com)
(PROTOCOL=TCP)(HOST=spot)(PORT=1599))
OK (290 msec)
OK (100 msec)
OK (70 msec)
OK (70 msec)
OK (60 msec)
OK (70 msec)
OK (70 msec)
OK (80 msec)
OK (180 msec)
OK (340 msec)
```

Below is an example of TNSPING attempting to connect to an invalid service name:

```
tnsping invalid_service_name
```

This attempt produces the following message:

```
TNS Ping Utility for SunOS:  Version 2.2.2.0.0 – Production on
10–Mar–95 10:10:58
Copyright (c) Oracle Corporation 1995.  All rights reserved.
TNS–03505: Failed to resolve name
```

Following is an example of using TNSPING to connect to a name that is valid, but that resolves to an address where no server is located (for example, the server may be down or not started):

```
tnsping testing
```

The following message is returned:

```
TNS Ping Utility for SunOS: Version 2.2.2.0.0 – Production on
10–Mar–95 10:11:42
Copyright (c) Oracle Corporation 1995.  All rights reserved.
```

```
Attempting to contact
(ADDRESS=(PROTOCOL=tcp)(HOST=spot)(PORT=1599))
  TNS-12541: TNS:no listener
```

Test All Client Types

Make sure that each client type is tested. It is not enough to test that the Interchange works. If there are several different client types in your network, initiate a connection to a server from each of them. If a connection is unsuccessful, use logging and tracing—the diagnostic tools—to find the cause of the problem. An error stack in the error log may point to the problem. If not, turn on tracing and repeat the operation. You can find information about error logging and tracing in Chapters 1 through 3 of the *Oracle Network Products Troubleshooting Guide.*

Diagnostics

Common Errors During Testing

If you are unsuccessful in bringing up a listener, Interchange, or Names Server, or fail to make a connection to a database, check to see if the cause is one of the following common errors.

- An invalid listener name was typed in the LSNRCTL START command.

  Check your typing. Check the LISTENER.ORA file to ensure that the listener name you are using is valid.

- Files were put in the wrong place.

  Both the listener and the Interchange will indicate that they cannot start because configuration files could not be found. Check your operating system-specific documentation to see that the LISTENER.ORA file has been placed correctly on the listener machine, and the INTCHG.ORA, TNSNAV.ORA, and TNSNET.ORA files are placed correctly on the Interchange machine. (Refer to "Testing the Network Configuration" in this chapter, and to the *Oracle Network Products Troubleshooting Guide* for more information.)

OS Doc

Diagnostics

  If you are using TNSNAMES.ORA files, make sure they were created with Network Manager, and not created by hand. If you are using Oracle Names Servers, make sure they have been started. Also make sure that NAMES.ORA is placed correctly, and that SQLNET.ORA file contains the preferred Names Server parameter. If a native naming service such as NIS is in use, make sure that the appropriate Native Naming adapter has been installed on clients and servers, and that service names have been

OS Doc

properly loaded into it. Refer to your operating system–specific documentation for information.

- Address already in use.

  Another process may already be using the address listed in LISTENER.ORA. On some protocols such as TCP/IP, DECnet, and OSI, each network service on a node must use a unique port or socket. On other network protocols such as SPX/IPX or NetBIOS, each network service name must be unique for the entire network. Another network service may be using the same configuration. Contact your network administrator to evaluate whether the network address is available.

- When trying to connect to a database, you may get the message ORA–12203: ”TNS:Unable to connect to destination” .

  Use the LSNRCTL utility to start the listener on the server machine. See Appendix C for further information.

- When trying to make a connection from a client, you may get the message ORA–12154:“TNS:Could not resolve service name”.

  The service name you requested is not listed in the TNSNAMES.ORA file, or the TNSNAMES.ORA file has been placed incorrectly. See Appendix C for further information.

- When trying to connect to a database, you may get the message ORA–1034: ”Oracle Not Available”.

  The database is not running on the server machine. A listener alone does not provide a database connection; the database instance must also be started.

- Connect requests that come in too quickly for a listener to handle, and which exceed the listener’s backlog (determined by QUEUESIZE parameter in LISTENER.ORA, TNSNET.ORA, and NAMES.ORA), are returned with an ECONNREFUSED error. A client encountering this error returns the message ”ORA–12541: No Listener” and the client log or trace files will show the ECONNREFUSED message.

  To correct this problem, follow these steps:

  – Stop the listener.

  – Reconfigure QUEUESIZE in LISTENER.ORA, TNSNET.ORA, or NAMES.ORA to be a larger value (based on anticipated simultaneous connect requests).

  – Restart the listener.

– Try to connect again.

Refer to ”Increasing the Queue Size for the Listeners” in Appendix A for information on increasing the number of connection requests the listener can handle.

Note: The default for QUEUESIZE is operating system-specific. Similarly, it is dependent upon the operating system whether the QUEUESIZE parameter will be recognized and be put into effect.

- When attempting to stop the listener, you may get the message TNS–01169: “The listener has not recognized the password.” Enter the SET PASSWORD command from within LSNRCTL, and then enter the STOP command to stop the listener.

Diagnostics

Other common errors are listed in Appendix C of this manual. All the error messages generated by SQL*Net, the Interchange, Oracle Names, Oracle Native Naming Adapters, and Oracle Network Manager and their underlying layers can be found in the *Oracle Network Products Troubleshooting Guide*. The *Oracle Network Products Troubleshooting Guide* also contains information about how to interpret log files and how to use the trace facility for troubleshooting purposes.

Using the Listener Control Utility

The network listener establishes listen endpoints on a machine. These listen endpoints are well–known addresses that clients and servers use to connect to a database. The Listener Control Utility, LSNRCTL, is a tool that you run from the operating system prompt to start and control the listener. For SQL*Net use, the general form of the Listener Control Utility is:

```
LSNRCTL command [listener_name]
```

In this syntax:

LSNRCTL	Specifies the name of the tool that controls the listener. In some operating systems, this fixed parameter is case sensitive. If the operating system is case sensitive, enter LSNRCTL in lowercase.
command	Specifies the action to be performed.
listener_name	Specifies the listener name defined in LISTENER.ORA. If *listener_name* is not included, the default value LISTENER is used.

You can also issue Listener Control commands at the program prompt. When you enter LSNRCTL on the command line, the program is opened. You can then enter the desired commands from the program prompt. For example, the following command starts the database subagent for a node managed by SNMP support.

```
LSNRCTL> dbsnmp_start
```

Commands for LSNRCTL Information

The Listener Control Utility has three commands that provide information about LSNRCTL itself. These commands are HELP, SET, and SHOW.

HELP

HELP provides a list of all the LSNRCTL commands available. An example follows:

```
iris[331] lsnrctl

LSNRCTL for SunOS: Version 2.3.1.1.0 – Beta on 17–AUG–95 17:15:02

Copyright (c) Oracle Corporation 1994.  All rights reserved.

Welcome to LSNRCTL, type ”help” for information.

LSNRCTL> help
The following operations are available
An asterisk (*) denotes a modifier or extended command:

start               stop                status
services            version             reload
trace               spawn               dbsnmp_start
dbsnmp_stop         dbsnmp_status       change_password
quit                exit                set*
show*
```

SET

This command lists the operations that can be set using the SET command.

```
LSNRCTL> set
The following operations are available after set
An asterisk (*) denotes a modifier or extended command:

password             trc_file             trc_directory
trc_level            log_file             log_directory
log_status           current_listener     connect_timeout
startup_waittime     use_plugandplay
```

SHOW

This command lists the operations that can be set using the SHOW command.

```
LSNRCTL> show
The following operations are available after show
An asterisk (*) denotes a modifier or extended command:

trc_file             trc_directory        trc_level
log_file             log_directory        log_status
current_listener     connect_timeout      startup_waittime
snmp_visible         use_ckpfile          use_plugandplay
```

LSNRCTL Commands to Control the Listener

The following commands act on the listener. If there is more than one listener on the node, the command acts against the default LISTENER, unless you provide another listener name on the command line.

CHANGE_PASSWORD [*listener_name*]

You can dynamically change the password of a listener using LSNRCTL. From within LSNRCTL, enter the following:

```
change_password
```

The control utility prompts you for your old password, then for the new one. It asks you to re-enter the new one, then changes it.

> **Note:** Neither the old nor the new password displays during this procedure.

The following is an example of changing a password:

```
iris> lsnrctl

LSNRCTL for SunOS: Version 2.3.2.0.0 – on 10–Sep–95 18:59:34
Copyright (c) Oracle Corporation 1994. All rights reserved.
Welcome to LSNRCTL, type ”help” for information.
LSNRCTL> change_password
Old password:
New password:
Reenter new password:
Connecting to (ADDRESS=(PROTOCOL=ipc)(KEY=iris))
Password changed for LISTENER
The command completed successfully
LSNRCTL>
```

SPAWN

Use the SPAWN command to start a program stored on the machine on which the listener runs, and which is listed with an alias in the LISTENER.ORA file.

For example, you might create a program called NSTEST, and manually add the following to the LISTENER.ORA file:

```
nstest =
(PROGRAM=(NAME=nstest)(ARGS=test1)(ENVS='ORACLE_HOME=/usr/oracle')
)
```

To run the NSTEST program, you would enter the following:

```
lsnrctl spawn nstest
```

START [*listener_name*]

Starts the named listener. If no listener name is entered, LISTENER is started by default.

STOP [*listener_name*]

Stops the named listener. If no listener name is entered, LISTENER is stopped by default.

> **Note:** You must have set a valid password if one is listed in the LISTENER.ORA parameter PASSWORDS_*listener_name* to be able to use this command.

STATUS [*listener_name*]

Displays basic information: version, start time, uptime, what LISTENER.ORA file is used, and whether tracing is turned on.

SERVICES [*listener_name*]

Provides detailed information about the services the listener listens for: how many connections have been established, how many refused. It displays three different types of services: dedicated servers from LISTENER.ORA, dispatcher information, and prespawned shadows.

> **Note:** You must have set a valid password if one is listed in the LISTENER.ORA paramete,r PASSWORDS_*listener_name*, to be able to use this command.

RELOAD [*listener_name*]

Shuts down everything except listener addresses, and re–reads the LISTENER.ORA file. This command enables you to add or change services without rebooting the system.

> **Note:** You must have set a valid password, if one is listed in the LISTENER.ORA file parameter PASSWORDS_*listener_name,* to be able to use this command.

TRACE [*listener_name*] *level*

Turns on tracing for the listener. Choices of level are OFF, USER, or ADMIN. USER provides a limited level of tracing; ADMIN provides a more detailed trace. This command overrides the setting in the LISTENER.ORA file. (This command has the same functionality as SET TRC_LEVEL.)

> **Note:** You must have set a valid password, if one is listed in the LISTENER.ORA file parameter PASSWORDS_*listener_name,* to be able to use this command.

Diagnostics

For detailed information on how to use tracing, see the *Oracle Network Products Troubleshooting Guide*.

VERSION [*listener_name*]

Use this command if you are asked to provide version information to Oracle Worldwide Customer Support. It provides version numbers for the following:

- listener executable version number
- TNS version number
- protocol adapter version numbers

DBSNMP_START

Use this command to start the SNMP subagent for an Oracle database running on the same node.

DBSNMP START must be run locally—you cannot run it remotely.

DBSNMP_STOP

Use this command to stop the SNMP subagent for an Oracle database running on the same node.

DBSNMP STOP must be run locally—you cannot run it remotely.

DBSNMP_STATUS

Use this command to verify whether the SNMP subagent for an Oracle database is running.

DBSNMP STATUS must be run on the same node the Oracle database is on.

SET PASSWORD *[listener_name]*

Enter this command if you want to perform administrator–only tasks on the listener. For example, you must enter the SET PASSWORD command before you can stop the listener. The password should match one listed in the LISTENER.ORA file. You may enter this command when you start up the shell or any time during your session.

The preferred, secure way to enter your password is in interactive mode. Enter the command from within LSNRCTL, for example,

```
LSNRCTL> SET PASSWORD
```

The Listener Control Utility responds:

```
enter listener password:
```

When you enter your password and press [Return], the password is not echoed on the terminal. You receive the message:

```
Command successful
```

Note: You must enter the SET PASSWORD command before you can stop the listener (with the STOP *[listener_name)*

Note: The listener supports encrypted and unencrypted passwords.

SET TRC_LEVEL [*listener_name*] *level*

Turns on tracing for the listener. Choices of level are OFF, USER, or ADMIN. Selecting USER provides a limited level of tracing; ADMIN provides a more detailed trace. This command overrides the setting in the LISTENER.ORA file. (This command has the same functionality as TRACE.)

Note: You must have set a valid password, if one is listed in the LISTENER.ORA file parameter PASSWORDS_*listener_name* to be able to use this command.

Diagnostics

For detailed information on how to use tracing, see the *Oracle Network Products Troubleshooting Guide.*

SET TRC_FILE [*listener_name*]

Use this command to set a non–default name for the trace file.

For example, the following command sets the name of the file that contains listener trace information:

```
LSNRCTL> set trc_file list.trc
```

The computer output would be something like the following:

```
Connecting to (ADDRESS=(PROTOCOL=ipc)(KEY=iris))
LISTENER parameter ”trc_file” set to list.trc
The command completed successfully
```

SET TRC_DIRECTORY [*listener_name*]

Use this command to set a non–default location for the trace file or to return the location to the default.

For example, the following command sets the directory in which the trace file is placed:

```
LSNRCTL> set trc_directory /usr/oracle/admin
```

The computer output would be something like the following:

```
Connecting to (ADDRESS=(PROTOCOL=ipc)(KEY=iris))
LISTENER parameter ”trc_directory” set to /usr/oracle/admin
The command completed successfully
```

SET LOG_STATUS

Logging for a listener is always ON. This command has no effect.

SET LOG_FILE [*listener_name*]

Use this command to set a non–default name for the log file.

For example, the following command sets the name of the file that contains listener log information:

```
LSNRCTL> set log_file list.trc
```

The computer output would be something like the following:

```
Connecting to (ADDRESS=(PROTOCOL=ipc)(KEY=iris))
LISTENER parameter ”log_file” set to list.log
The command completed successfully
```

SET LOG_DIRECTORY [*listener_name*]

Use this command to set a non–default location for the log file or to return the location to the default.

For example, the following command sets the directory in which the log file is placed:

```
LSNRCTL> set log_directory /usr/oracle/admin
```

The computer output would be something like the following:

```
Connecting to (ADDRESS=(PROTOCOL=ipc)(KEY=iris))
LISTENER parameter "log_directory" set to /usr/oracle/admin
The command completed successfully
```

SET CURRENT_LISTENER [*listener_name*]

If there is more than one listener on a node, any LSNRCTL command acts on the default listener (LISTENER) unless another listener has been set.

For example, suppose there were two listeners on a node, LISTENER and LSNR1. If you wanted to set or show parameters for LSNR1, you would first need to send the following command from within LSNRCTL:

```
LSNRCTL> set current_listener lsnr1
```

Any subsequent LSNRCTL commands within the same LSNRCTL session would then apply to LSNR1, unless CURRENT_LISTENER were reset. For example, if the current listener had been set to LSNR1, then the STAT command would produce something like the following output:

```
LSNRCTL> stat
Connecting to (ADDRESS=(PROTOCOL=IPC)(KEY=IRIS))
STATUS of the LISTENER
------------------------
Alias                     lsnr1
Version                   TNSLSNR for SunOS: Version 2.3.1.1.0– Beta
Start Date                18–Aug–95 11:25:45
Uptime                    0 days 0 hr. 0 min. 3 sec
Trace Level               admin
Security                  OFF
SNMP                      ON
Listener Parameter File   /etc/oracle/network/admin/listener.ora
Listenr Log File          /etc/oracle/network/log/lsnr1.log
Listener Trace File       /etc/oracle/network/trace/lsnr1.trc
Services Summary...
```

```
  db1      has 1 service handler(s)
The command completed successfully
```

You can also display the current listener by using the LSNRCTL SHOW command.

> **Note:** You must enter SET CURRENT_LISTENER from within the LSNRCTL utility. When you exit the utility, the setting will be lost.

SET CONNECT_TIMEOUT TIME [*listener_name*]

This command determines the amount of time the listener will wait for a valid connection request after a connection has been started

```
LSNRCTL> set connect_timeout 20
```

The computer output would be something like the following:

```
Connecting to (ADDRESS=(PROTOCOL=ipc)(KEY=iris))
LISTENER parameter ″connect_timeout″ set to 20
The command completed successfully
```

SET STARTUP_WAITTIME TIME [*listener_name*]

This command sets the amount of time the listener sleeps before responding to a STATUS command:

```
lSNRCTL> set startup_waittime 10
```

The computer output would be something like the following:

```
Connecting to (ADDRESS=(PROTOCOL=ipc)(KEY=iris))
LISTENER parameter ″startup_waittime″ set to 10
The command completed successfully
```

SET USE_PLUGANDPLAY [*listener_name*] ON | OFF

Use this command to determine whether the Dynamic Discovery Option of Oracle Names is enabled on a listener:

```
LSNRCTL> set use_plugandplay ON|OFF
```

By default, the value of this parameter is OFF. If Oracle Names version 2 has been installed, and you want the listener to use the Dynamic Discovery Option (that is, to register itself with a well–known Names Server, set it to ON. You can also set the value in LISTENER.ORA through Oracle Network Manager, which places a slightly different parameter into LISTENER.ORA, as follows:

```
use_plugandplay_listener_name=[ON|OFF]
```

If the Dynamic Discovery Option is enabled, you can use the LSNRCTL STATUS command to see whether a service has registered itself. For example:

```
LSNRCTL> stat lsnr1
Connecting to (ADDRESS=(PROTOCOL=IPC)(KEY=IRIS))
STATUS of the LISTENER
------------------------
Alias                     lsnr1
Version                   TNSLSNR for SunOS: Version 2.3.1.1.0– Beta
Start Date                18–Aug–95 11:25:45
Uptime                    0 days 0 hr. 1 min. 49 sec
Trace Level               admin
Security                  OFF
SNMP                      ON
Listener Parameter File /etc/oracle/network/admin/listener.ora
Listener Log File         /etc/oracle/network/log/lsnr1.log
Listener Trace File       /etc/oracle/network/trace/lsnr1.trc
Services Summary...
  db1 (Registered)        has 1 service handler(s)
The command completed successfully
```

SHOW [*listener_name*] *subcommand*

All of the SET commands listed except SET PASSWORD have equivalent SHOW commands. In response to one of the SHOW comands, LSNRCTL displays the current setting of the listener for that parameter.

In addition, there are two other parameters that can be shown, but not set, through LSNRCTL.

SHOW [*listener_name*] SNMP_VISIBLE

This command displays whether the listener is accessible to SNMP clients:

```
LSNRCTL> show snmp_visible
```

The computer output would be something like the following:

```
Connecting to (ADDRESS=(PROTOCOL=ipc)(KEY=iris))
LISTENER parameter ”snmp_visible” set to ON
The command completed successfully
```

> **Note:** The SNMP_VISIBLE parameter can be displayed, but not set, through LSNRCTL:

SHOW [*listener_name*] USE_CKPFILE

Use LSNRCTL to see whether the the use of a checkpoint file has been enabled in LISTENER.ORA:

```
LSNRCTL> show use_ckpfile
```

The computer output would be something like the following:

```
Connecting to (ADDRESS=(PROTOCOL=ipc)(KEY=iris))
LISTENER parameter "use_ckpfile" set to OFF
The command completed successfully
```

If USE_CKPFILE_*listener_name* is not set in LISTENER.ORA, and you use LSNRCTL SET commands to change listener parameters, when you stop the listener, LSNRCTL sends a message reminding you that the changes you have made are not persistent.

> **Note:** You cannot set this parameter using LSNRCTL; it must be set in LISTENER.ORA. For more information about the use of checkpoint files, see "Persistent Changes" later in this chapter.

QUIT

Use this command to quit LSNRCTL and return to the operating system prompt. (Same as EXIT.)

EXIT

Use this command tp quit LSNRCTL and return to the operating system prompt. (Same as QUIT.)

Making Changes Persistent

If set to ON, the following parameter in LISTENER.ORA will make the changes you make by using the SET command in LSNRCTL persistent; that is, if you shut down the listener and then start it up again, the parameters will retain the values you set through LSNRCTL.

```
use_ckpfile_listener_name=ON
```

If this parameter is in the LISTENER.ORA file, the parameters you set using LSNRCTL are saved to a checkpoint file, *listener_name*.CKP, either in the TNS_ADMIN directory if one exists, or in the *ORACLE_HOME*/NETWORK/ADMIN directory. The values in this checkpoint file will override the values set in LISTENER.ORA.

If the USE_CKPFILE parameter is set to OFF in LISTENER.ORA, the values set by LSNRCTL are lost when the listener is restarted or the LISTENER.ORA parameters are reloaded. The default is for USE_CKPFILE to be OFF.

Note: If checkpointing is enabled and parameters in LISTENER.ORA are changed, the changes will not be visible because they are overridden by the parameters in the checkpoint file. If you want the changes in LISTENER.ORA to be visible, set the value of USE_CKPFILE to OFF and use the LSNRCTL RELOAD command to reread the LISTENER.ORA file.

Examples of the Use of LSNRCTL

This section provides some examples of the most common uses of the LSNRCTL utility.

Starting the Listener

To start a default listener, that is, a listener defined in the LISTENER.ORA file using the name LISTENER, use the command:

```
LSNRCTL START
```

Alternatively, you can enter LSNRCTL on the command line and then enter START from the program prompt.

To start a listener configured in the LISTENER.ORA file with a name other than LISTENER, include that name. For example, if the listener name is TCP_LSNR, enter:

```
LSNRCTL START TCP_LSNR
```

Or, from the LSNRCTL program prompt, just enter:

```
LSNRCTL> START TCP_LSNR
```

Stopping the Listener

To stop the default listener, use the command:

```
LSNRCTL STOP
```

To stop a running listener defined in LISTENER .ORA as TCP_LSNR, use the command:

```
LSNRCTL STOP TCP_LSNR
```

Remember, if there are any passwords in the LISTENER.ORA file, you must use the SET PASSWORD command before you can use the STOP command. You must set the password from within the LSNRCTL program; you cannot set it from the operating system command line. The method for setting the password depends on whether you are using the encrypted password feature. If you are *not* using an encrypted password, enter the password on the LSNRCTL command line. For example, the following commands stop the TCP_LSNR using an unencrypted password:

```
LSNRCTL
LSNRCTL> SET PASSWORD password
LSNRCTL> STOP TCP_LSNR
```

If you are using an encrypted password, enter the password in interactive mode. For example, the following commands stop the listener named TCP_LSNR:

```
LSNRCTL
LSNRCTL> SET PASSWORD
   Enter listener password (password is not displayed)
   Command successful
LSNRCTL> STOP TCP_LSNR
```

Stopping a listener when in batch mode is not recommended, because to do so you must include your password in a cleartext batch file, which would threaten your security. However, if you are not using an encrypted password, stopping a listener can be done by redirecting input into the command interpreter.

> **Note:** You should not stop the listener in batch mode if it requires an encrypted password.

Different operating systems use different syntax. An example for VMS and two alternative methods for UNIX follow.

To stop a listener in batch mode on VMS, create a DCL script, with a name like LSNRSTOP.COM, as follows:

```
$ lsnrctl
set password password
stop listener_name
exit
```

When you want to stop the listener, run the script as follows:

```
@LSNRSTOP
```

On a UNIX system or on OS/2, the following procedure would be effective:

Create a file with a name like LSNRSTOP that contains the following lines:

```
set password password
stop listener_name
```

You can then stop the listener from the command line by entering:

```
lsnrctl < LSNRSTOP
```

Alternatively, in UNIX you can stop the listener by creating a shell script named something like ”lsnrstop”. The shell script would look something like this::

```
lsnrctl <<!
set password password
stop listener_name
exit
!
```

You can then stop the listener from the command line by entering:

```
LSNRSTOP
```

Again, Oracle Corporation recommends against stopping a listener in batch mode because of the need to expose your password.

> **Note:** Be careful when stopping a listener. On some platforms and with some protocols, when a listener is stopped any SQL*Net connections currently running are shut down. In some situations the connections continue, but it is then not possible to start the listener again until the running processes have been closed. It is good practice to send a warning message to all network users before stopping a listener.

Checking Listener Status

One of the most useful commands available with the Listener Control Utility is the STATUS command. With the status command, an administrator can get a view of the "state" of a listener by checking:

Diagnostics

- the current setting of the logging and tracing options. These are described in detail in the *Oracle Network Products Troubleshooting Guide*.
- the list of database SIDs available through this listener. These are defined in the SID mapping in LISTENER.ORA.
- whether a password is encrypted in LISTENER.ORA. (If you encrypt the listener password you can have only one password.)
- whether the network listener can respond to queries from an SNMP–based network management system
- the address the TNSLSNR is listening on

Suppose that a user wanted to know the status of a listener on a UNIX system defined in LISTENER.ORA with the default name LISTENER. The user would enter the following command at the operating system command line:

```
LSNRCTL STATUS
```

Following is sample output from a status report of a UNIX listener :

```
LSNRCTL for SunOS: Version 2.2.2.0 – Production on 15–FEB–95
07:07:10
Copyright (c) Oracle Corporation 1995. All rights reserved.
```

```
Connecting to (ADDRESS=(PROTOCOL=IPC)(HOST=orchid)(port=1334))
STATUS of the LISTENER
------------------------
Alias                     LISTENER
Version                   TNSLSNR for SunOS: Version 2.2.2.0 –
  Production
Start Date                10–FEB–94 07:06:34
Uptime                    0 days 0 hr. 0 min. 44 sec
Trace Level               ADMIN
Security                  ON
SNMP                      ON
Listener Parameter File
/private1/dvl/7012/network/admin/listener.ora
Listener Log File
/private1/dvl/7012/network/log/listener.log
Listener Trace File
/private1/dvl/7012/network/trace/listener.trc
Listening on              (ADDRESS=(PROTOCOL=tcp)(HOST=orchid)
  (port=1334)))
                          (ADDRESS=(PROTOCOL=decnet)(node=23.106)
  (object=orchid)))
Services Summary...
  orchid           has 1 service handlers
The command completed successfully
```

Retrieving Listener Services

A database administrator who wanted to get information about the services of the listener would enter the following commands from within LSNRCTL:

```
LSNRCTL>[SET PASSWORD password]
LSNRCTL> SERVICES
```

The output of a LSNRCTL SERVICES command follows:

```
LSNRCTL for SunOS: Version 2.1.3.0.0 – Production on 10–FEB–94
07:14:55

Copyright (c) Oracle Corporation 1993.  All rights reserved.

Connecting to (ADDRESS=(PROTOCOL=IPC)(KEY=ruth))
Services Summary...
  ruth             has 1 service handlers
    DEDICATED SERVER established:99 refused:0
The LSNRCTL command completed successfully
```

In this example, the LSNRCTL SERVICES command returned the information that the listener had established 99 connections using a dedicated server process and refused none.

In the following example the LSNRCTL SERVICES command returns information about four types of service handlers.

```
LSNRCTL for SunOS: Version 2.2.2.0.0 – Production on 15–Mar–1995 17:41:12

Copyright (c) Oracle Corporation 1994.  All rights reserved.

Connecting to (ADDRESS=(PROTOCOL=ipc)(KEY=orchid))
Services Summary...
   listener          has 4 service handlers
    DEDICATED SERVER established:0 refused:0
    PRESPAWNED SERVER established:0 refused:0 current:0 max:1 state:ready
      PID:15439
      (ADDRESS=(PROTOCOL=ipc)(DEV=4)(KEY=#15439.1))
    PRESPAWNED SERVER established:5 refused:0 current:0 max:1 state:ready
      PID:15441
      (ADDRESS=(PROTOCOL=tcp)(DEV=4)(HOST=139.185.22.25)(PORT=3334))
    DISPATCHER established:30 refused:0 current:7 max:21 state:ready
      D000 (machine: orchid, pid: 15406)
      (ADDRESS=(PROTOCOL=tcp)(DEV=7)(HOST=139.185.22.25)(PORT=3330))
The command completed successfully
```

This message shows that since this listener has been running, it has made a total of five connections using a prespawned server for TCP/IP, and that it has one running at this time. It also shows that there are seven connections currently established using a multi-threaded server dispatcher, with a total of 30 established since the process started.

Modifying Client Parameters

Client configuration parameters are contained in the SQLNET.ORA file. Generally, you generate SQLNET.ORA files for clients with similar navigation needs (called *client profiles* or *client types*) using Oracle Network Manager. The SQLNET.ORA file determines the following client characteristics:

- preferred choices of name respolution services
- preferred Names Servers (if Oracle Names is used on the network)
- preferred Oracle MultiProtocol Interchanges (if used on the network)
- non–default tracing and logging characteristics
- preferred security services (if Secure Network Services is used on the network)

- characteristics of connection requests (whether to use a dedicated server, out-of-band breaks, IPC addresses)
- client registration information for statistical reporting

The SQLNET.ORA file created by Network Manager generally is used by a number of clients. If you want to change any of these parameters for a whole group of clients, you may use Network Manager to change the network definition and create new SQLNET.ORA files to be distributed as needed.

Alternatively, if you want to change parameters for individual clients, you may use the SQLNET.ORA Editor. This utility is part of the Client Status Monitor. It enables you to open the SQLNET.ORA file on an individual client and change its parameters either through a graphical user interface or through the command line.

If you are using Oracle Names version 2.0 and the Dynamic Discovery Option, you do not necessarily need to use Network Manager. If you do not use it, a SQLNET.ORA file is not created. If you find you want to provide some non–default parameters for the client, you can create an empty SQLNET.ORA using any text editor, such as vi, and then add the parameters you need using the SQLNET.ORA Editor. Using this method to create the file makes it less likely that typographical or formatting mistakes will cause errors.

Controlling Network Services from Network Manager

Oracle Network Manager version 3.0 lets you run some Listener Control Utility, Names Control Utility, and Interchange Control Utility commands without having to go to a command line prompt to execute them.

After your network has been defined, you can obtain information on any network service by selecting it on the Map View or Tree View Object List and then selecting Control Network Service from the Special menu.

> **Note:** You must define the network configuration and distribute the configuration files throughout the network before you can use this option.

A control window appears. Select a command from the drop-down list. (The commands available depend on the type of network object you have selected.) For example, if you select Trace, select the level of tracing you want from the Level list. The result of your command will be displayed in the large read–only box on this window.

Following are the commands you can use from Network Manager to control network services remotely.

LSNRCTL	VERSION STATUS RELOAD SET_TRACE_LEVEL SERVICES
NAMESCTL	VERSION STATUS PING SET TRACE_LEVEL SHOW TRACE_LEVEL
INTCTL	VERSION STATUS

ONames

Intchg

The Listener Control Utility (LSNRCTL) commands are described in "Using the Listener Control Utility" in this chapter. See the *Oracle Names Administrator's Guide* for information on NAMESCTL commands. See the *Oracle MultiProtocol Interchange Administrator's Guide* for information on INTCTL commands.

Initiating a SQL*Net Connection

There are a number of ways to initiate a connection with an Oracle server. Commonly used methods are:

- the operating system command line
- a tool logon screen
- a 3GL program
- special commands within certain tools

The specifics of use are slightly different in each case. Each of the general methods listed is briefly covered here. To identify the method used in a specific tool, refer to the tool's user's guide.

Connecting from the Operating System Command Line

The general form of connecting an application to a database server from the command line is:

```
tool username/password@service_name
```

In this syntax:

tool Specifies the command used to invoke a tool such as SQL*Plus, SQL*Forms, etc.

username Specifies an Oracle username on the server.

password Specifies the corresponding password on the server.

service_name Specifies a service name entered in the TNSNAMES.ORA file that identifies the connect descriptor for the desired server. If the server is in the client's default domain, the service name does not need to include the domain name. However, if the server is in another domain, the service name must include the domain. (The default domain is determined by a parameter in the client's SQLNET.ORA file. See the section on the SQLNET.ORA file in Appendix A of this manual.)

For example, in a network with only one domain, the default .WORLD domain, it is not necessary to include .WORLD in the service name. For example:

```
% sqlplus scott/tiger@SERVERX
```

However, if the client's default domain were .EAST and the server's domain were .WEST, then the service name would have to include the domain. For example,

```
% sqlplus scott/tiger@SERVERX.WEST
```

Note: To prevent the password from displaying during a logon, you can leave out the password parameter on the command line; you will then be prompted to enter your password without it showing on screen.

Most Oracle tools can use the operating system command line to connect; some provide alternatives.

Connecting from the Tool Logon Screen

Some tools provide a logon screen as an alternative form of logon. A user can log on to a database server just as easily by identifying both the username and service name in the username field of the tool logon screen, and typing the password as usual in the password field. Figure 5 – 1 shows a SQL*Forms logon screen where the user SCOTT is connecting to the server SERVERX with a password of TIGER. Notice the password cannot be seen, a standard feature of Oracle tool logon screens.

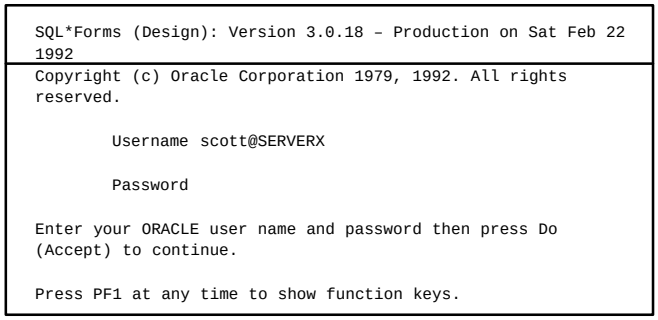

Figure 5 – 1 Connection from Logon Screen

Connecting from a 3GL Application

In applications written using a 3GL, the program must establish a connection to a server using the following syntax:

```
EXEC SQL CONNECT :username IDENTIFIED BY :password
```

In this connection request, the :*username* and :*password* are 3GL variables that can be set within the program either statically or by prompting the user. When connecting to a database server, the value of the :*username* variable is in the form:

username@service_name

which is the same as in the tool logon screen above. The :*password* variable contains the password for the database account being connected to.

Connecting Using Special Commands within Tools

Some Oracle tools have commands for database connection, once the tool has been started, to allow an alternative username to be specified without leaving the tool. Both SQL*Plus and SQL*DBA allow the CONNECT command using the following syntax:

```
SQL> CONNECT username/password@service_name
```

For example:

```
SQL> CONNECT SCOTT/TIGER@SERVERX
```

This is very similar to the operating system command line method, except that it is entered in response to the tool prompt instead of the operating system prompt.

Other Oracle tools use slightly different methods specific to their function or interface. For example, Oracle CDE tools use logon buttons and a pop–up window with the username, password, and remote database ID field. For more information on connecting to Oracle with a specific tool, refer to the tool's user guide.

Distributed Database Management

SQL*Net plays a major role in distributed database management. In a distributed transaction, SQL*Net provides the means for clients and servers to communicate by way of their SQL–based dialog language. SQL*Net performs transparently to enable distributed database functions. This section highlights the database functionality used in conjunction with SQL*Net.

Database Links

A *database link* is a database object that links an Oracle account in one database to an Oracle account in another database. The data in the remote schema is then accessible to the database in which the database link is defined.

> **Note:** If your network uses Oracle Names, a global database link is created on every database on the network to every other database. Therefore, you do not need to create additional database links, as described in this section. See ”Database Links wth Oracle Names” later in this chapter.

The generic syntax for creating a database link in SQL is:

```
CREATE [PUBLIC] DATABASE LINK linkname
[CONNECT TO username IDENTIFIED BY password]
USING 'service_name'
```

In this syntax:

[PUBLIC]	Specifies a database link available to all users with the CREATE SESSION privilege. If the PUBLIC option is omitted, a private link available only to the creator is created. Note that creating a public database link requires CREATE PUBLIC DATABASE LINK privilege.
linkname	Specifies the name of the database link. If the remote server is in the local server’s domain, the link name does not need to include the domain name. However, if the server is in another domain, the link name must include the domain. (The domain is determined by DB_DOMAIN in the initialization parameter file).
CONNECT TO	Optionally specifies a single username and password for all users of the database link to share. If the clause is omitted, the Oracle username and password of the user account using the database link will be used to connect to the remote database server.

username Specifies a valid Oracle username on the remote database server.

password Specifies the corresponding password of the username on the remote database server.

service_name Specifies the service name defined in the TNSNAMES.ORA file or stored in Oracle Names associated with the connect descriptor for the desired database. If the remote server is in the local server's default domain, the service name does not need to include the domain name. However, if the server is in another domain, the service name must include the domain. (The default domain is determined by a parameter in the server's SQLNET.ORA file. See the section on the SQLNET.ORA file in Appendix A of this manual.)

Prior to Oracle7, a database administrator could specify any linkname for a database link. However, with Oracle7 and later, a database link must have the same name as the global database name of the database. Remember that the service name is also the same as the global database name; therefore, the linkname and service name are now the same.

For example, the command for creating a public database link to a database which has the global database name ORCHID.HQ.ACME is as follows:

```
CREATE PUBLIC DATABASE LINK ORCHID.HQ.ACME
CONNECT TO scott IDENTIFIED BY tiger
USING 'ORCHID.HQ.ACME'
```

> **Note:** The CONNECT TO *username* IDENTIFIED BY *password* clause and the USING '*global_database_name*' clause are both optional.

Public Database Links with a Default Connection

Figure 5 – 2 shows a public database link created by the DBA user SYSTEM using the service name NY_FIN.HQ.ACME. The link is created by entering:

```
CREATE PUBLIC DATABASE LINK NY_FIN.HQ.ACME
USING 'NY_FIN.HQ.ACME'
```

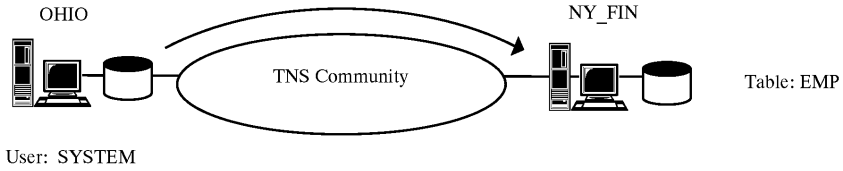

Figure 5 – 2 Public Database Link with Default Connection

Users connected to OHIO.SALES.ACME can use the NY_FIN.HQ.ACME database link to connect to NY_FIN.HQ.ACME with the same username and password they have on OHIO.SALES.ACME. To access the table on NY_FIN.HQ.ACME called EMP, any user could issue the SQL query:

```
SQL> SELECT * FROM EMP@NY_FIN.HQ.ACME;
```

> **Note:** If the target database were in the default domain of the source database, the user would not need to include the domain in the link name or service name, or in the SELECT command.

This query would initiate a connection from OHIO to NY_FIN using the current username and password to log onto NY_FIN. The query would then be processed on NY_FIN, and the data available to the current user from the table EMP would be returned to OHIO. Each user creates a separate connection to the server. Subsequent queries to that database link by that user would not require an additional logon.

Public Database Links with a Specific Connection

Figure 5 – 3 shows the database link created by the user SYSTEM with the service name NY_FIN:

```
CREATE PUBLIC DATABASE LINK NY_FIN
CONNECT TO FINPUBLIC IDENTIFIED BY NOPASS
USING 'NY_FIN'
```

> **Note:** The CONNECT TO *username* IDENTIFIED BY *password* clause and the USING '*global_database_name*' clause are both optional.

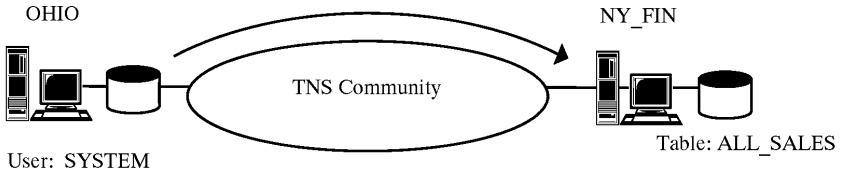

Figure 5 – 3 Public Database Link with Specific Connection

Any user connected to OHIO can use the NY_FIN database link to connect to NY_FIN with the common username / password of FINPUBLIC / NOPASS. To access the table in the FINPUBLIC account of NY_FIN called ALL_SALES, any user could issue the SQL query:

```
SQL> SELECT * FROM ALL_SALES@NY_FIN;
```

This query would initiate a connection from OHIO to NY_FIN to the common account FINPUBLIC. The query would be processed on NY_FIN and data from the table ALL_SALES would be returned to OHIO.

Each user creates a separate connection to the common account on the server. Subsequent queries to that database link by that user would not require an additional logon.

Connection Qualifiers

You can also define *connection qualifiers* to database links. Connection qualifiers provide a way to create more than one link to a given database. Alternate links are a useful way to access different accounts on the same database with different sets of access privileges. The alternate link created by a connection qualifier must include a reference to a database by its global database name (or service name).

A connection qualifier contains a qualifier name and, optionally, a username and password. To create a connection qualifier, use a statement similar to the following:

```
CREATE PUBLIC DATABASE LINK NY_FIN@PROFITS
CONNECT TO ACCOUNTS IDENTIFIED BY TAXES
USING 'NY_FIN'
```

To use the connection qualifier, you append the qualifier name to the service name of the database you want to access.

For example, the following SQL queries use three separate database links to the same database, using different connection qualifiers:

```
SELECT * FROM EMP@NY_FIN;
SELECT * FROM SCHEDULE@NY_FIN@PROFITS;
SELECT * FROM EMPSALARIES@NY_FIN@FIN;
```

In this example @PROFITS and @FIN are connection qualifiers.

Dropping a Database Link

You can drop a database link just as you can drop a table or view. The command syntax is:

```
DROP DATABASE LINK linkname;
```

For example, to drop the database link NY_FIN, the command would be:

```
DROP DATABASE LINK NY_FIN;
```

Finding Available Database Links

Server

Any user can query the data dictionary to determine what database links are available to that user. For information on viewing the data dictionary, refer to the *Oracle7 Server Concepts* or the *Oracle7 Administrator's Guide.*

Database Links with Oracle Names

When you define a network that includes Oracle Names, Network Manager automatically creates a global database link to every database server you define from every other database server in the network. These database links do not reside in the data dictionary, but in the network definition to which the Names Servers refer. The database links thus created do not initially include a CONNECT TO clause, so that users reach the linked database using the same usernames and passwords as they use to reach the first database.

```
SQL> SELECT * FROM EMP@OHIO, DEPT@NY_FIN;
```

Explicitly Defined Database Links

NetMan

You can edit global database links to include CONNECT TO data using Network Manager. When you edit a database, you can specify a single default username and password for the database link. See Chapter 5 in the *Oracle Network Manager Administrator's Guide* for details on how to edit database links.

Connection Qualifiers

NetMan

You can also define *connection qualifiers* to global database links through Network Manager. Connection qualifiers provide a way to create multiple links to the same database. Multiple database links to the same database provide different access routes with different accounts and privileges. See Chapter 5 in the *Oracle Network Manager Administrator's Guide* for details on how to create connection qualifiers using the Network Manager.

ONames

For a more detailed discussion of database links, see *Oracle7 Server Distributed Systems, Volume I,* and the *Oracle Names Administrator's Guide.*

Synonyms

Database *synonyms* are a standard SQL feature used to provide alternate names for database objects and, optionally, their locations. A synonym can be created for any table, view, snapshot, sequence, procedure, function, or package. All synonyms are stored in the data dictionary of the database in which they are created. To simplify remote table access through database links, a synonym can be defined to allow single–word access to remote data, isolating the specific object name and the location from users of the synonym. The syntax to create a synonym is:

```
CREATE [PUBLIC] SYNONYM_name
FOR [schema.]object_name[@database_link_name]
```

In this syntax:

[PUBLIC]	Specifies that this synonym is available to all users. Omitting this parameter makes a synonym private, and usable only by the creator. Public synonyms can be created only by a user with CREATE PUBLIC SYNONYM system privilege.
synonym_name	Specifies the alternate object name to be referenced by users and applications.
schema	Specifies the schema of the object specified in object_name. Omitting this parameter uses the creator's schema as the schema of the object.
object_name	Specifies either a table, view, sequence, or other name as appropriate.
database_link_name	Specifies the database link which identifies the remote username in which the object specified in object_name is located.

A synonym must be a uniquely named object for its schema. If a schema contains a database object and a public synonym exists with the same name, Oracle always finds the database object when the user that owns the schema references that name.

Because a synonym is merely a reference to the actual object, the security domain of the object is used when the synonym is accessed. For example, a user that has access to a synonym for a specific table must also have privileges on that table to access the data in it. If the user attempts to access the synonym, but does not have privileges on the table it identifies, an error occurs indicating that the table or view does not exist.

Figure 5 – 4 shows two servers, OHIO and NY_FIN, in which a database link from OHIO to NY_FIN and the synonym FOR_SALE provide an

alternate object name for use in OHIO to reference the OPEN table in NY_FIN. The database link and the synonym are created as follows:

```
CREATE PUBLIC DATABASE LINK NY_FIN
CONNECT TO REAL_ESTATE IDENTIFIED BY NOPASS;
USING 'NY_FIN'
CREATE PUBLIC SYNONYM FOR_SALE
FOR OPEN@NY_FIN;
```

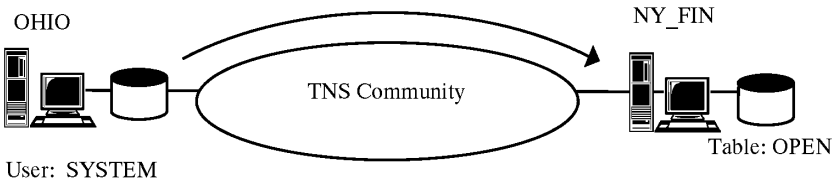

Figure 5 – 4 Using Synonyms for Alternate Object Names

The table OPEN on NY_FIN could be accessed from OHIO using the SQL statement:

```
SELECT * FROM FOR_SALE;
```

Using this database link, the user is logging on to NY_FIN as user REAL_ESTATE. Notice that this public synonym was created by the DBA on behalf of the REAL_ESTATE username. If the table OPEN were owned by another user, such as SALES_MGR, the CREATE SYNONYM statement would have referred to the object as SALES_MGR.OPEN@NY_FIN. Without such a prefix, a table that does not exist in the database link user's schema would return an error, since it would be looking for the OPEN table owned by the REAL_ESTATE user.

Maintaining Location Transparency

When using a synonym to access a database object over a database link, the user of the synonym is said to have *location transparency*. For example, an application developer using the FOR_SALE synonym from the previous example has the illusion that FOR_SALE is a database object available for use as any other object would be. The reference to the database link is invisible to the developer; therefore, any application built using the synonym would have no reference to the location of specific data.

This ability to isolate applications from the location of data in a distributed transaction ensures maximum flexibility for future enhancements or changes to the application. For example, if the OPEN table were to move from one database server to another, only the synonym or the database link would need to be changed to identify the

new location. The applications would continue to reference the same object name, although they would be connecting to a new location to access the data in that table. Figure 5 – 5 shows the most common method of redefining the location of a table to retain location transparency.

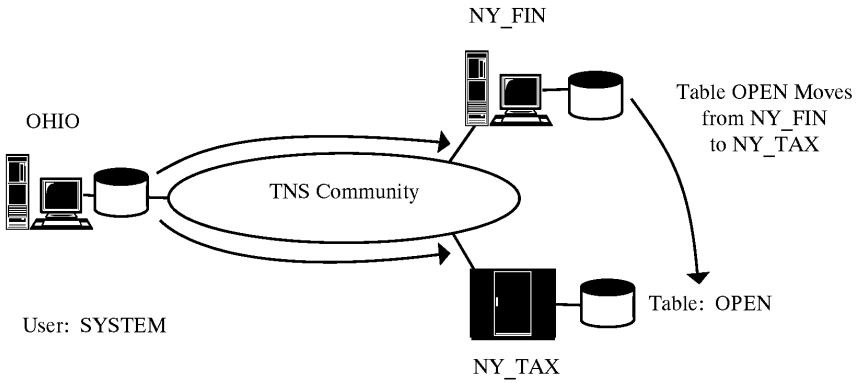

Figure 5 – 5 Redefining Table Location to Retain Location Transparency

```
CREATE PUBLIC DATABASE LINK NY_TAX
CONNECT TO REALTOR IDENTIFIED BY NOPASS
USING 'NY_TAX'
DROP SYNONYM FOR_SALE;
CREATE PUBLIC SYNONYM FOR_SALE
FOR OPEN@NY_TAX;
```

To relocate the table, a second database link was created called NY_TAX that connected to a new database with the service name NY_TAX, and the synonym was recreated to reference the NY_TAX database link instead of the NY_FIN database link. Any other tables that were accessed through the NY_FIN database link to NY_FIN would continue to function properly.

Alternatively, if the only table being accessed on NY_FIN were the OPEN table, the synonym could have remained unchanged and the database link redefined to use the service name NY_TAX instead of NY_FIN. Either option is equally effective.

Snapshots

An Oracle system with both the distributed option and the procedural option can replicate tables that are frequently queried by users on many nodes of a distributed database. By having read–only copies of heavily accessed data on several nodes, the distributed database does not need to send information across a network repeatedly, thus helping to improve the performance of the database application. Oracle provides an automatic method for table replication called *snapshots*. Snapshots are

read–only copies of a master table located on a remote node. A snapshot can be queried, but not updated; only the master table can be updated. A snapshot is periodically refreshed to reflect changes made to the master table.

Maintaining snapshots of a master table among the nodes of a distributed database is often a useful feature for the following reasons:

- Queries can be issued against a local snapshot. Therefore, associated query performance is fast because the requested data does not have to be shipped over a network.
- If the database that contains the master table or a database that contains a snapshot experiences a failure or is cut off from the distributed database by a network failure, the other (read–only) copies remain accessible to other users of the system.

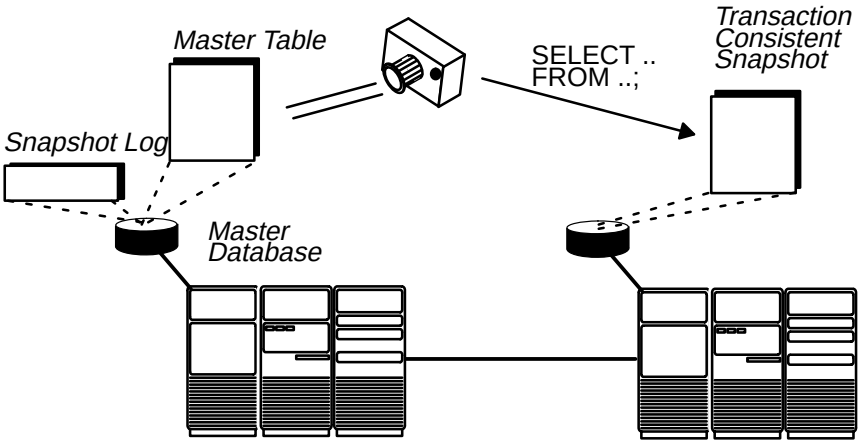

Figure 5 – 6 Table Replication Using Snapshots

A snapshot is a full copy or a subset of a table that reflects a recent state of the master table. A snapshot is defined by a distributed query that references one or more master tables, view, or other snapshots. A database that contains a master table is referred to as the master database.

Simple vs. Complex Snapshots Each row in a simple snapshot is based on a single row in a single remote table. Therefore, a simple snapshot's defining query has no GROUP BY or CONNECT BY clauses, or subqueries, joins, or set operations. If a snapshot's defining query contains any of these clauses or operations, it is referred to as a *complex snapshot*.

Creating Snapshots

Create a local snapshot using the SQL command CREATE SNAPSHOT. As when creating tables, you can specify storage characteristics for the snapshot's data blocks, extent sizes and allocation, and the tablespace to hold the snapshot; or you can specify a cluster to hold the snapshot. Unique to snapshots, you can specify how the snapshot is to be refreshed and the distributed query that defines the snapshot. You must fully qualify any remote table names used in the defining query. For example, the following CREATE SNAPSHOT statement defines a local snapshot to replicate the remote EMP table located in the SCOTT schema in NY:

```
CREATE SNAPSHOT emp_sf
   PCTFREE 5 PCTUSED 60
   TABLESPACE users
   STORAGE (INITIAL 50K NEXT 50K PCTINCREASE 50)
   REFRESH FAST
           START WITH sysdate
           NEXT sysdate + 7
   AS SELECT * FROM scott.emp@ny;
```

Whenever a snapshot is created, it is immediately populated with the rows returned by the query that defines the snapshot. Thereafter, the snapshot is refreshed as specified by the REFRESH clause; see *Oracle7 Server Distributed Systems, Volume 2: Replicated Data* for more information about refreshing snapshots.

When a snapshot is created, Oracle creates several internal objects in the schema of the snapshot. These objects should not be altered. At the snapshot node, Oracle creates a *base table* to store the rows retrieved by the snapshot's defining query. Oracle then creates a read–only view of this table that is used whenever queries are issued against the snapshot.

Specifying the Defining Query of a Snapshot

The defining query of a snapshot can be any valid query of tables, views, or other snapshots that are not owned by user SYS. The query cannot contain either an ORDER BY or FOR UPDATE clause. Furthermore, simple snapshots are defined using a query that does not contain GROUP BY or CONNECT BY clauses, or join, subquery, or set operations.

The query that defines a snapshot can define a snapshot with a different structure from that of the master table. For example, the following CREATE SNAPSHOT statement creates a local snapshot named EMP_DALLAS, with only the EMPNO, ENAME, and MGR columns of the master table (in New York), and only the rows of the employees in department 10:

```
CREATE SNAPSHOT emp_dallas
   .
   .
   .
   AS SELECT empno, ename, mgr
      FROM scott.emp@ny
      WHERE deptno = 10;
```

Privileges Required to Create Snapshots

To create a snapshot, the following sets of privileges must be granted as follows:

- To create a snapshot in your own schema, you must have the CREATE SNAPSHOT, CREATE TABLE, and CREATE VIEW (only for simple snapshots) system privileges as well as the appropriate SELECT privilege on the master tables.
- To create a snapshot in another user's schema, you must have the CREATE ANY SNAPSHOT, CREATE ANY TABLE, CREATE ANY VIEW, and CREATE ANY INDEX (only for simple snapshots) system privileges as well as the appropriate SELECT privilege on the master table. Additionally, for the snapshot to be of any use, the owner of the snapshot must have the appropriate SELECT privilege on the master tables

In both of the above cases, the owner of the snapshot must also have sufficient quota on the tablespace intended to hold the snapshot.

The large set of privileges required to create a snapshot is due to the underlying objects that must also be created on behalf of the snapshot.

Refreshes

Periodically, a snapshot is refreshed to reflect the current state of its master table. To refresh a snapshot, the snapshot's defining query is issued and its results are stored in the snapshot, replacing the previous snapshot data. Each snapshot is refreshed in a separate transaction, either automatically by Oracle (according to the interval set when the snapshot was defined or altered) or manually. Instructions for refreshing table snapshots are included in the *Oracle7 Server Administrator's Guide*.

Snapshot Logs

A simple snapshot can be refreshed from a snapshot log to expedite the refresh process. A *snapshot log* is a table in the master database that is

associated with the master table. Oracle uses a snapshot log to track the rows that have been updated in the master table when a certain simple snapshot based on the master table is refreshed. One snapshot log can be used by multiple simple snapshots). Only the appropriate rows in the snapshot log need to be applied to the snapshot to refresh it (called a *fast refresh*). If no other simple snapshot requires an applied row in the log, it is purged from the log to keep the log size small; however, if another simple snapshot requires the row for its next refresh, the row remains in the log.

A complex snapshot or simple snapshot without a snapshot log must be completely regenerated using the master tables every time the snapshot is refreshed (called a *complete refresh*).

Creating Snapshot Logs

Snapshot logs are created in the master database using the SQL command CREATE SNAPSHOT LOG. You can set storage options for the snapshot log data blocks, extent sizes and allocation, and tablespace to hold the snapshot log. The following statement creates a snapshot log associated with the EMP table:

```
CREATE SNAPSHOT LOG ON emp
   TABLESPACE users
   STORAGE (INITIAL 10K NEXT 10K PCTINCREASE 50);
```

The snapshot log is always created in the same schema that contains the master table. Since you cannot specify a name for the snapshot log (one is implicitly given by Oracle), uniqueness is not a concern.

If you own the master table, you can create an associated snapshot log if you have the CREATE TABLE and CREATE TRIGGER system privileges. If you are creating a snapshot log for a table in another user's schema, you must have the CREATE ANY TABLE and CREATE ANY TRIGGER system privileges. In either case, the owner of the snapshot log must have sufficient quota in the tablespace intended to hold the snapshot log.

The privileges required to create a snapshot log directly relate to the privileges necessary to create the underlying objects associated with a snapshot log.

Using Snapshots

Snapshots are queried just like a table or view. For example, the following statement queries a snapshot named EMP:

```
SELECT * FROM emp;
```

To query a snapshot, you must have the SELECT object privilege for the snapshot, either explicitly or via a role.

In release 7.0 of the Oracle Server, snapshots are read–only. You cannot issue any INSERT, UPDATE, or DELETE statements when using a snapshot; if you do, an error is returned. Although INSERT, UPDATE, and DELETE statements can be issued against the base table for a snapshot, they can corrupt the snapshot. Never manipulate data in the base table of a snapshot. Updates are allowed on the master table only, which must then be refreshed to update the snapshot.

Creating Views and Synonyms Based on Snapshots

Views or synonyms can be defined based on snapshots. The following statement creates a view based on the EMP snapshot:

```
CREATE VIEW sales_dept AS
   SELECT ename, empno
   FROM emp
   WHERE deptno = 10;
```

Dropping Snapshots

You can drop a snapshot independently of the master tables or a snapshot log if you decide that you no longer want to replicate a table in a database. To drop a local snapshot, use the SQL command DROP SNAPSHOT. For example:

```
DROP SNAPSHOT emp;
```

If you drop the only snapshot of a master table, you should also drop the snapshot log of the master table, if appropriate.

Only the owner of a snapshot, or a user with the DROP ANY SNAPSHOT, DROP ANY TABLE, and DROP ANY VIEW system privileges can drop a snapshot.

For more information about table replication, snapshots, and distributed database design, refer to the following manuals: *Oracle7 Server Distributed Systems, Volumes I and II*, and the *Oracle7 Server Administrator's Guide.*

Copying Data between Databases

When working with a distributed database, you may want to move data from one database to another. For example, you may want to download sales data from a particular region to your workstation for further analysis. The SQL*Plus COPY command lets you copy data from:

- a remote database to your local database
- your local database to a remote database
- one table on a remote database to another table on that database

- a table on one remote database to a table on another database

You do not need to be directly connected to any of the databases in order to copy the information from one to another. These operations, except the first, are impossible to perform with database links and queries. Using the COPY command is similar to using the CREATE DATABASE LINK command, in that you must know:

- a valid service name for the remote database
- a valid username and password for the remote database

The COPY command syntax is:

```
COPY FROM username/password@service_name
[TO username/password@service_name]
(APPEND | CREATE | INSERT | REPLACE)
tablename [columnlist]
USING subquery
```

In this syntax:

service_name	Specifies a valid service name in the TNSNAMES.ORA file. Each service name maps to a connect descriptor that connects to a database using the keyword SID.
APPEND	Specifies that if the destination table already exists, COPY will insert the copied data into it. If the table does not already exist, COPY will create it and then insert the copied data into it.
CREATE	Specifies that COPY will create a new table and insert the copied data into it. If the table already exists, COPY will report an error and stop processing.
INSERT	Specifies that if the destination table already exists, COPY will insert the data into it. If the table does not already exist, COPY will report an error and stop processing.
REPLACE	Specifies that if the table already exists, it will be dropped and replaced with the new data. If the table does not already exist, it will be created and the data inserted into it.
tablename	Specifies the destination table to which you are copying data.

columnlist	Specifies that only data from the specified columns should be copied.
subquery	Specifies the query to be used to select the data to be copied.

Note that you can use any kind of query to select the data to be copied. For example, you can COPY select rows from the source table by using a WHERE clause. You can also COPY specific columns from the source table by listing those columns after the SELECT statement, instead of using the asterisk that means all columns. The copied columns will have the same names in the destination table that they had in the source table.

Following is an example of a COPY command that copies only two columns from the source table specified by the alias BOSTON, and copies only those rows in which the value of DEPTNO is 30. Since the TO clause is omitted, the table is copied to the database to which SQL*Plus is currently connected:

```
COPY FROM SCOTT/TIGER@BOSTON
     REPLACE EMPCOPY2
     USING SELECT ENAME, SAL
          FROM EMP
          WHERE DEPTNO = 30;
```

In the following example, the command copies the entire table EMP from the remote database specified by the SALES service name in the TNSNAMES.ORA file. Because the TO clause is omitted, the table is copied to the database to which SQL*Plus is currently connected. The new table is named EMP2:

```
COPY FROM SCOTT/TIGER@SALES
     CREATE EMP2
     USING SELECT * FROM EMP;
```

You can also copy data to a local table from a remote table using the syntax:

```
COPY FROM SCOTT/TIGER@servicename
     INSERT INTO local_table
     SELECT * FROM remote_table@dblink
```

For further information about the COPY command, see the *SQL*Plus User's Guide and Reference Manual.*

Support for Operating System Authorized Logins

In SQL*Net version 1, Oracle provided support for operating system authorized logins across a network using the special database login accounts. This support continues in SQL*Net version 2 on the same platforms and protocols for compatibility of function, but is not expanded to additional platforms and protocols.

Operating System Authorized Logins

Operating system authorized logins to a database allow a user who has operating system privileges on a machine to get automatic access to an Oracle database on the machine without supplying a database password. The assumption is that having supplied a password to the operating system, that user can be trusted to access his data in his own operating–system–authorized user account on that machine. The default prefix for this account is OPS$; however, the database administrator may assign some other prefix.

Extending this notion over a network, operating system authorized logins allow users to log into a database on a remote machine using their local userid, without supplying a database password. The operating system user account must match the username of the operating system authorized account.

For further information about user authentication, see the *Oracle7 Server Distributed Systems, Volume I*.

Warning about Operating System Authorized Logins

It is not uncommon for many employees to have DOS or other types of workstations on their desks over which they have complete control. It is a trivial task for such a person to create a username of any valid string of characters. If John Doe is the president of a company and uses operating system-authorized logon accounts across the network, any user can create a JDOE account on a personal workstation and access John's data without a password. On some server platforms this is more difficult than described, but never impossible. This is a particularly serious problem where an OPS$ account has DBA privileges. In such cases, it is very easy to masquerade as an Oracle DBA.

Oracle's Recommendation Regarding Operating System Authorized Logins

Because of these security implications, Oracle Corporation recommends against using operating system authorized logins to remote databases. This does not mean you cannot use operating system authorized login accounts in your database; it simply prevents them from being used across the network. The same OPS$ account can still be accessed if a password is supplied in connections over the network.

If you have applications that rely on operating system authorized logins across a network, you should consider altering them to require the user to supply a password when connecting over the network.

In SQL*Net version 1 the ability to perform operating system authorized logins exists in some SQL*Net TCP/IP and SQL*Net DECnet drivers on UNIX and VMS systems. The default setting is for operating system authorized logins to be enabled unless explicitly disabled. We recommend that you explicitly disable operating system authorized logons where they exist in any version 1 drivers you have installed. For information regarding your platform and SQL*Net Version 1 drivers, and how operating system authorized logins can be disabled, see your Oracle operating system specific manuals regarding SQL*Net version 1.

OS Doc

In SQL*Net version 2, operating system authorized login support is enhanced to make it more difficult to act as an imposter, but it is still not impossible. By default, operating system authorized logins are disabled in SQL*Net version 2.

> **Note:** If externally authenticated logins are enabled (by setting the REMOTE_OS_AUTHENT parameter to TRUE), it will be possible to connect without a password even though the underlying SQL*Net protocol cannot verify the identity of the user. For this reason, REMOTE_OS_AUTHENT should always be set to FALSE (the default value).
>
> If it is necessary to make externally authenticated logins, then a secure protocol such as the Bequeath (Pipe) driver should be used.

In all cases with SQL*Net version 1 or version 2, Oracle recommends leaving remote DBA privileges disabled.

Support for Network Authentication Adapters in Release 2.2

With SQL*Net release 2.2 and Oracle7 Server release 7.2, the ability to use network authentication services for single sign–on and smart card authentication is provided. Use Oracle Network Manager to configure these authentication adapters. A site will have a choice of which adapters to link into their SQL*Net configuration. The Secure Network Services version 2 product is required to enable these authentication adapters. For information on configuring network authentication adapters, see the *Secure Network Services Administrator's Guide.*

SNS

A discussion of security issues in networked systems is in Chapter 6 of *Oracle7 Server Distributed Systems, Volume I.*

Authentication Services

Authentication services for single sign–on servers or smart cards are usually part of a Network Operating System (NOS) which overlays

several machines. Administration of a network with many machines can be centralized by creating ”network users” who have the same identity and privileges no matter which machine they are actually using.

Network Identity

All authentication services provide the concept of a network identity so that no matter which machine a user is logged into, he or she can log onto the network and be identified as the same person. This method of authentication ensures that a user is who he or she claims to be.

Network Roles

Because most operating systems support some concept of roles, groups, or identifiers, many authentication services such as single sign–on servers based on Kerberos can support the concept of network roles. For example, an administrator creates the role and then assigns it to certain network users. Role definitions are consistent across all machines linked by the authentication service.

Secure External Authentication Logins

Using the network identity provided by the authentication service, the database can provide secure external authentication over a non–secure protocol such as TCP / IP.

> **Note:** *External authentication* refers to authentication by either a network service or the operating system. *OS authentication* refers to operating system authorization only.

Users request external authentication using the same method they used previous to release 7.2. Users requesting external authentication still use a ’/’ to indicate the lack of a username and password.

```
SVRMGR> CONNECT /@ny
Connected.
```

If an authentication adapter is available, the server will use it to find the user’s network identity. If an authentication adapter is not available, operating system authorization will be performed. If the connection is not secure, the value of REMOTE_OS_AUTHENT will be used to determine if the login may proceed.

> **Note:** It is highly recommended that REMOTE_OS_AUTHENT be set to FALSE, because most protocols are not secure.

Secure Database Links

Proxy authentication (secure database links, that is, secure server-to-server connections) are used when a user who has been externally authenticated by a network service attempts to use an anonymous database link (one without a username or password specified). However, the network service must support some type of credential that can make use of a proxy login. When a connection is requested, the credentials are passed from the local database server to

the remote server. The remote server uses the credentials to obtain the network identity of the originating client process.

Network Roles and Privileges

Similar to the secure authentication behavior, if an authentication adapter is linked in, any external roles are retrieved from the network and not the operating system. If an authentication adapter is available (installed and linked into SQL*Net configuration), then it is used to authenticate the user; if not, then the operating system is used. If network roles are supported by the authentication service, they are used.

Configuring external roles varies depending on the authentication service. Instead of using the SID to name network roles (as OS roles does), network roles use the global database name. For example, for DCE authentication, something similar to the following naming convention to configure roles would be used:

```
ORA_<global_database_name>_<role_name>[_[A][D]]
```

> **Note:** Exact syntax to define network roles and privileges may vary depending on the authentication adapter in use.

Remote authentication provides a network version of the OSDBA and OSOPER privileges: SYSDBA and SYSOPER refer to the privileges necessary to perform an internal connection, whether verified by the operating system, password file, or network. A format similar to the following would be used to define network privileges:

```
ORA_<global_database_name>_[DBA|OPER]_SYS
```

SNS

See *Secure Network Services Administrator's Guide* for information on configuring network roles and privileges for specific authentication adapters.

CHAPTER

6 Configuring Oracle SNMP Support

This chapter provides an overview of the Oracle SNMP Support feature, available with Oracle7 Server release 7.2 and SQL*Net release 2.2 and later. Oracle SNMP Support enables Oracle products running anywhere on an enterprise's network to be located, identified, and monitored by a management station running at one or more centrally located nodes.

NetMan

OS Doc

SNMP

This chapter briefly describes the tasks necessary to configure Oracle SNMP Support for the Oracle7 Server, Listener, MultiProtocol Interchange, and Oracle Names. For specifics on how to configure SNMP support, see *Oracle Network Manager Administrator's Guide*. For specific instructions on how to install and configure SNMP support on a particular platform see your operating system–specific documentation.

For a complete description of SNMP concepts and terms, see the *Oracle SNMP Support Reference Guide*. It provides detailed listings of what the management information contains and suggestions on how to use it effectively in developing SNMP–based management applications.

What is the Purpose of SNMP?

The Simple Network Management Protocol (SNMP) was designed for the purpose of querying and transferring management information between nodes on a network. As a protocol, SNMP defines how information is conveyed between ”managed” nodes and ”managing” nodes on an enterprise’s network. In the past, managed nodes have included network devices such as bridges, routers, hubs, network interface cards, network monitors, and analyzers. However, the set of managed network objects is evolving to include software applications in addition to network devices. The managing nodes consist of workstations at which any one of a number of system and network management applications are run. These programs, based upon SNMP, are variously called *network management platforms* or *frameworks.*

SNMP

For a discussion of terms, concepts, and principal components related to SNMP, such as network management station (”manager nodes”), managed nodes or elements, Management Information Base (MIB), and SNMP Master Agent and subagent, refer to the *Oracle SNMP Support Reference Guide.*

Oracle SNMP Support for Oracle Services

Oracle SNMP Support is a bundled feature of SQL*Net release 2.2. SNMP is a standard underlying many popular network management systems.

Oracle SNMP support allows an Oracle7 Server, listener, MultiProtocol Interchange, or Oracle Names Server to be queried by any SNMP–based network management system such as Hewlett-Packard’s OpenView, IBM’s NetView6000, and DEC’s POLYCENTER / NetView. SNMP support is built into the SQL*Net listener, Oracle Names, and MultiProtocol Interchange—it is not a separate executable. For example, when a network listener, Oracle Names Server, or MultiProtocol Interchange is started, SNMP support is automatically started. The SNMP Master Agent must be running before starting any subagent. For information on starting the SNMP Master Agent refer to your operating system–specific documentation.

The SNMP subagent for the Oracle7 Server is a separate executable, and must be started separately. For information see ”Controlling theOracle Database Subagent” later in this chapter.

Benefits of Oracle SNMP Support

Oracle SNMP support allows a network listener, Oracle7 database, Oracle Names Server, and MultiProtocol Interchange to be queried by any SNMP–based network management system. The primary benefits of Oracle SNMP Support are:

- Oracle products can be located, identified, and monitored in realtime across enterprise networks of any size.
- Database administrators see the current status of Oracle products, selecting from among several status variables that are defined for each product in a management information base (MIB).

By using a management framework such as HP's OpenView, DBAs can see at a glance how the Oracle environment is running. Among the tasks that Oracle SNMP Support allows DBAs or network administrators to do are to:

- monitor the status of Oracle services such as databases or tools throughout the network
- identify performance bottlenecks
- "discover" Oracle databases or tools as they start up on any node
- receive alerts when exceptional events occur, for instance, the database going down
- define thresholds and automatic responses to specific events
- detect and diagnose potential problems quickly and easily
- be notified when certain events occur
- store, report upon, filter, and analyze historical data

Installing and Configuring Oracle SNMP Support

This section briefly describes the major tasks that the network administrator must perform to enable SNMP support for an Oracle network service.

Consider Hardware and Software Requirements

There are certain hardware and software requirements that need to be met. See your platform–specific documentation for more information.

Installing Oracle SNMP Support

Oracle SNMP Support is installed with the Installer. For information on how to install the product on your platform refer to the operating system–specific documentation.

Configuring SNMP Support

Following are some general tasks the network administrator needs to perform to configure SNMP support on a managed node:

> **Note:** The actual file locations and utilities may vary from platform to platform.

1. Configure the SNMP Master Agent. Refer to your operating system-specific documentation for instructions.
2. Configure the subagent for the Oracle7 server, listener, MultiProtocol Interchange, and Oracle Names network services using the SNMP pages in Network Manager.

   To configure an SNMP subagent, you configure between three and five of the following parameters on the Database SNMP, Listener SNMP, Names Server SNMP, and MultiProtocol SNMP Pages:

   - SNMP Visible
   - SNMP Index
   - SNMP Contact
   - Username—optional (specified for an Oracle database server only)
   - Password—optional (specified for an Oracle database server only)

NetMan

   For instructions on configuring SNMP support for Oracle services with Network Manager, see Chapter 5, ”Entering Component Information” in the *Oracle Network Manager Administrator’s Guide.*

   After creating the network definition and configuring SNMP support, you generate configuration files for the network, including an SNMP.ORA file for each managed node. An SNMP.ORA file is stored in the same directory as the other SQL*Net configuration files. For example, on most UNIX platforms, the directory location is $ORACLE_HOME/network/admin. (Refer to ”Oracle SNMP Support Configuration File” and ”SNMP.ORA” in Appendix A for an example and information on the SNMP.ORA file.)

   Attention: Make sure that you assign each visible service a unique index number.

3. Run the DBSNMP.SQL script to create the new user, role, and grants required by Oracle SNMP Support in all databases visible over SNMP.

If you change the default username and password with the DBSNMP.SQL script, you must also specify the new username and password in the SNMP page of Network Manager.

Using Oracle SNMP Support

Following are things you must do before using Oracle SNMP Support:

Start the Master Agent, Encapsulator, and Native SNMP Agent

This varies depending on the platform. See your operating system-specific documentation for the actual command names and instructions.

Controlling the Oracle Database Subagent

The SNMP subagent is a separate executable from the database, and it is not started automatically when the database is started.

To start, stop, and check to see if the database subagent is running or not, use the following commands from LSNRCTL:

```
LSNRCTL> dbsnmp_start
LSNRCTL> dbsnmp_stop
LSNRCTL> dbsnmp_status
```

You must run these commands on the machine where the SNMP subagent is running—there is no remote support.

Attention: The Master Agent must be running before starting the subagent.

Controlling SNMP Support for Network Services

The subagents for the listener, MultiProtocol Interchange, and Oracle Names Server are started when the listener, MultiProtocol Interchange, are Oracle Names Server are started. They are not separate executables, as with SNMP support for the Oracle database.

Polling Database MIB Variables

In using, tuning, and understanding applications that poll the SNMP-managed applications, some skill is required in taking the proper measurements at the proper times, accurately interpreting the data, and making sure that the very act of managing a process does not impose undue performance demands on that process.

Attention: Measuring performance should not negatively impact the performance.

As inherent components of the applications, the subagents for the listener, Interchange and Oracle Names always have access to the current value for any MIB variable. However, as a separate process from the database, there is some cost associated with the database subagent's

querying a value. To reduce the impact of its queries against the database, the database subagent maintains a cache of values that it has retrieved from the database. The length of time that each variable is considered ”fresh” may have one of two values: FAST or SLOW.

All variables retrieved by the database subagent, whether in the standard or Oracle Database MIB, are labeled as either FAST or SLOW. Only variables in the two database MIBs have this classification. Networking product subagents have access to internal data structures that are updated continuously; they do not cache values like the database Agent.

Polling Intervals

The polling times are hardcoded and cannot be configured. If polling times could be configured, problems in developing applications would result. Graphical elements would exhibit a false value, depending upon the frequency of polling and cache intervals. If various machines and services on a network had different polling intervals, significant problems might result.

FAST Variables

FAST variables are retrieved every 30 seconds or more. If a new query comes in and the subagent sees that the cached value is less than 30 seconds, it returns that value AGAIN. If the cached value is older than 30 seconds, it obtains the new value and returns it, as well as caches the new value.

SLOW Variables

SLOW variables are retrieved once every 5 minutes or more.

APPENDIX

A

Contents of the Configuration Files

This appendix describes the contents of the configuration files needed for SQL*Net release 2.3. It describes what components they are used by, their purpose, and their contents. The syntax of the files is described in Appendix B, "Syntax Rules for Configuration Files." However, because you create and maintain the configuration files using Network Manager, you probably do not need to be concerned about the details of their syntax.

This appendix provides a brief overview of all the configuration files for the Oracle network products, then describes the files for SQL*Net in detail. These files are:

- LISTENER.ORA
- TNSNAMES.ORA
- SQLNET.ORA
- PROTOCOL.ORA
- SNMP.ORA

Intchg

ONames

Configuration files for the related Oracle networking products, the MultiProtocol Interchange and Oracle Names, are not described in detail here. They are discussed in detail in the *Oracle MultiProtocol Interchange Administrator's Guide* and the *Oracle Names Administrator's Guide*, respectively.

Configuration files for the Oracle Server, such as INIT.ORA, are described in detail in the *Oracle7 Server Administrator's Guide*.

Overview of the Configuration Files

NetMan

OS Doc

This section provides a brief description of the configuration files for each component in the network. It includes the types of information required in each file, and shows the relationships between them. Information about the locations of these files is in Chapter 6, "Distributing the Configuration Files," of the *Oracle Network Manager Administrator's Guide* and in the Oracle operating system-specific documentation for your platform. Each file is described in more detail in later sections of this chapter.

Client Configuration Files

Clients typically have three configuration files that are created by Oracle Network Manager. These files provide information about the following:

- network destinations
- network navigation
- tracing and logging, and security (encryption and checksumming)

TNSNAMES.ORA — This file contains a list of the service names and addresses of network destinations. A client (or a server that is part of a distributed database) needs this file to tell it where it can make connections.

Note: This file is not necessary if Oracle Names is used.

Note: This file should be generated and modified by Oracle Network Manager. Do not edit it manually.

TNSNAV.ORA — This file is used only in a network that includes one or more Oracle MultiProtocol Interchanges. It lists the communities of which the client (or server) is a member and includes the names and addresses of the Interchanges available in local communities as a first hop toward destinations in other communities.

Note: This file should be generated and modified by the Oracle Network Manager. Do not edit it manually.

SQLNET.ORA — This file contains optional diagnostic parameters, client information about Oracle Names, and may contain other optional parameters such as native naming or security (encryption and checksumming) parameters.

Note: SQLNET.ORA may contain node–specific parameters. Unless you are using Oracle Names and the Dynamic Discovery Option, you should create it with Network Manager. You may edit the SQLNET.ORA file for an individual client by using the SQLNET.ORA Editor, which is described in the *Oracle Network Products Troubleshooting Guide.*

Diagnostics

In addition, clients and servers on some protocols may require PROTOCOL.ORA, which you must create manually.

PROTOCOL.ORA — This file contains protocol- and platform-specific options for protocols that require them, such as Async and APPC/LU6.2.

Server Configuration Files

Servers in a network that includes distributed databases also require the files that are needed by clients, because when servers connect to other database servers through database links they are, in effect, acting like clients.

In addition to the client configuration files described above, each server machine needs a LISTENER.ORA file to identify and control the behavior of the listeners that listen for the databases on the machine.

LISTENER.ORA — This file includes service names and addresses of all listeners on a machine, the system identifiers (SIDs) of the databases they listen for, and various control parameters used by the Listener Control Utility.

Note: Unless you are using Oracle Names and the Dynamic Discovery Option, this file should be generated and modified by the Oracle Network Manager. You should not edit it manually.

Note: LISTENER.ORA and TNSNAMES.ORA contain some similar information. The address of the server in TNSNAMES.ORA is the same as the address of the listener for a server in LISTENER.ORA. Similarly, the address in the TNSNAMES.ORA file includes the SID which is required (as SID_NAME) in the LISTENER.ORA file. Figure A – 1 shows the similarities between these files for a single server.

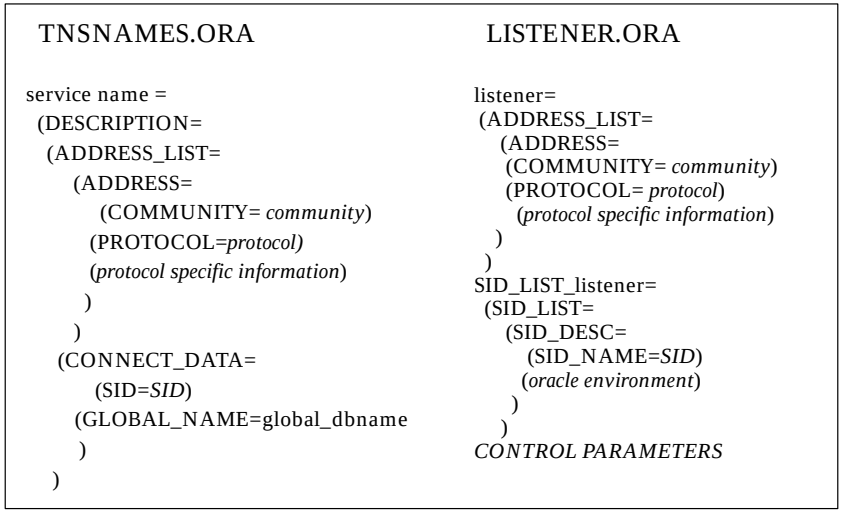

Figure A – 1 Similarities Between TNSNAMES.ORA and LISTENER.ORA

Interchange Configuration Files

Each Interchange in a network requires three configuration files. The files provide information about the following:

- network layout
- network navigation
- control parameters

TNSNET.ORA — This file contains a list of the communities in the network and the relative cost of traversing them, and the names and addresses of all the Interchanges in the network. This file provides an overview of the layout of the network for all the Interchanges.

TNSNAV.ORA — This file describes the communities of each individual Interchange on the network.

INTCHG.ORA — This file provides both required and optional parameters that control the behavior of each Interchange.

Note: These files should be generated and modified through Oracle Network Manager. They should not be edited by hand.

Intchg

For detailed information about the configuration files for the Interchange, see the *Oracle MultiProtocol Interchange Administrator's Guide*.

Oracle Names Configuration Files

ONames

Unless you are using the Dynamic Discovery Option, each Names Server in the network requires an individual configuration file called NAMES.ORA, as well as parameters in SQLNET.ORA.

For more information about the NAMES.ORA file for the Names Server, see Chapter 6 in the *Oracle Names Administrator's Guide.*

NAMES.ORA	Unless you are using the Dynamic Discovery Option, every node running a Names Server must have a NAMES.ORA file. NAMES.ORA contains control parameters for the Names Server and points the Names Server to the database where the network definition is stored. **Note:** This file should be generated and modified by Oracle Network Manager. Do not edit it manually.
SQLNET.ORA	This file contains client information about Oracle Names such as the default domain for service names stored in Oracle Names, and lists preferred Oracle Names Servers. It may also contain optional logging and tracing (diagnostic), native naming, and security (encryption, checksumming, and authentication) parameters.

Oracle SNMP Support Configuration File

Each node managed by Oracle SNMP Support requires a configuration file named SNMP.ORA. The parameters in SNMP.ORA:

- identify the services on the node that will be monitored by SNMP
- assign index numbers to the services that will be monitored
- contain contact information (name and phone number) for each service to be monitored
- specify a non–default time interval (in seconds) that the subagent polls an SNMP–managed database

ONames

For information on how to configure SNMP support so that the listener, Oracle7 Server, MultiProtocol Interchange, and Oracle Names Server can be queried by any SNMP–based network management system, see the *Oracle Network Manager Administrator's Guide* and the Oracle operating system-specific documentation.

> **Note:** You must generate the SNMP.ORA file with Network Manager.

SNMP

For detailed information about the Oracle SNMP Support feature and a description of its Management Information Base (MIB) variables, see the *Oracle SNMP Support Reference Guide.*

Configuring the Network Listener: LISTENER.ORA

Before a database server can receive connections from SQL*Net version 2 (and later) clients, a listener must be active on the server platform. On most platforms, a network listener is used. The configuration file for the network listener is LISTENER.ORA. It contains four parts:

- listener names
- listener addresses
- description of the databases that use the listener
- parameters that influence the listener's behavior

Note: This file should be generated and modified through the Oracle Network Manager. It should not be edited manually.

Listener Names

The listener name can be any easy–to–use name. The default listener name is LISTENER, which is the recommended name in a standard installation that requires only one listener on a machine. The listener name must be unique on the network. However, this uniqueness is assured by the fact that Network Manager appends the name of the node and its domain to the listener name you supply. For example, if there is a listener on a node named RACER and a listener on a node named RABBIT, Network Manager will append the node names and the domain to their names so that they will be identified as LISTENER_RACER.WORLD and LISTENER_RABBIT.WORLD.

The listener name must be unique to the machine. If you have more than one listener on a machine, each must have a unique name. The TURTLE node, for example, might have three listeners with the names LSNR1_TURTLE.WORLD, LSNR2_TURTLE.WORLD, and LSNR3_TURTLE.WORLD.

Listener Addresses

The listener usually listens both for internal connection requests and for connection requests from across the network.

IPC Addresses

The listener listens for interprocess calls (IPC) as well as calls from other nodes. For example, on a UNIX machine, an IPC adapter is the adapter for the UNIX domain socket communication mechanism; on VMS, it is the adapter for the mailbox communication mechanism. IPC addresses must be included in the LISTENER.ORA file. Oracle Network Manager generates the IPC entries automatically, without input from you.

The IPC address format, which is the same across platforms, is as follows:

```
(ADDRESS=
     (PROTOCOL=IPC)
     (KEY=string))
```

Network Manager creates two IPC addresses for each database for which a listener listens. In one, the key value is equal to the service name. This IPC address is used for connections from other applications on the same node. Service names are described later in this chapter, in the section ”TNSNAMES.ORA.” In the other IPC address, the key value is equal to the database system identifier (SID), which is described in the next section. This IPC address is used by the database dispatcher to identify the listener.

> **Note:** If the service name is the same as the SID, only one IPC address is needed, and only one is generated by Network Manager.

If the network includes Oracle Names, and if you create an alias (a second service name) for the address using Network Manager, an IPC address using the alias as a key is included in LISTENER.ORA.

Network Addresses

The network address of a listener includes the community in which the destination resides, the protocol it uses, and protocol–specific parameters. If the listener is on a node that is on more than one community, it has more than one address.

The Network Manager automatically provides the correct protocol specific parameters for any protocol you use, but you must provide the appropriate values. For information about the values for the parameters for a given protocol, see the Oracle operating system specific documentation for your platform.

OS Doc

Sample Addresses

Here is an example of an address for a listener on in a TCP/IP community:

```
LISTENER_mike.world=(ADDRESS_LIST=
          (ADDRESS=
                (PROTOCOL=IPC)
                (KEY=prod.world)
          )
          (ADDRESS=
                (PROTOCOL=IPC)
                (KEY=db1)
          )
          (ADDRESS=
```

```
                    (PROTOCOL=tcp)
                    (HOST=mike.world)
                    (PORT=1521)
               )
          )
```

Describing the Databases on the Listener

The next section of the LISTENER.ORA file describes the system identifiers (SIDs) of the databases for whom the listener listens. It is made up of keyword–value pairs.

```
SID_LIST_listener_name=[(SID_LIST=]
                    (SID_DESC
                       (GLOBAL_DBNAME=global_dbname)
                       (SID_NAME=SID)
                       (OS_Oracle_
                            environment=db_location)
                    )
                    [(SID_DESC=
                       (GLOBAL_DBNAME=global_dbname)
                       (SID_NAME=SID)
                       (OS_Oracle_environment=db_location)
                    )]
               [)]
```

OS Doc

The GLOBAL_DBNAME is the name and domain of the database as given in the database initialization parameter file. The *SID* is the Oracle system ID of the database server. In the next keyword–value pair, the keyword is operating system specific: it is indicated here as the variable *OS Oracle environment*. Its value, indicated here as *db_location*, is the specific location of the database executables.

An example typical of the UNIX operating system follows:

```
(ORACLE_HOME=/usr/oracle)
```

An example from a VMS environment follows:

```
(PROGRAM='disk$:[oracle.rdbms]orasrv.com')
```

An example for OS/2 might be:

```
(PROGRAM=ORACLE7)
```

An example of a complete SID_LIST_*listener_name* section on a UNIX operating system follows:

```
SID_LIST_LISTENER=(SID_LIST=
                    (SID_DESC=
                           (SID_NAME=db1)
                           (ORACLE_HOME=/usr/oracle)
                    )
```

```
                (SID_DESC=
                      (SID_NAME=db3)
                      (ORACLE_HOME=/usr/oracle)
                )
          )
```

Note that a listener can listen for more than one database on a machine. However, you may create different listeners for the databases if you wish. All the listeners on a single machine share one LISTENER.ORA file.

Prespawned Dedicated Server Processes

If you want the listener to create prespawned dedicated server processes when it is started, use Network Manager to include the following parameters in each SID_DESC in LISTENER.ORA.

PRESPAWN_ MAX	The maximum number of prespawned dedicated server processes the listener will create. This number must be at least as many as the sum of the pool size for each protocol. For greatest efficiency, Oracle Corporation recommends a large value, so that prespawned dedicated server processes are always available for new connection requests.
PROTOCOL	The protocol on which the listener creates prespawned dedicated server processes. If a listener listens on more than one community, you can choose whether to have pre–spawned servers on any or all of them.
POOL_SIZE	The number of unused prespawned dedicated server processes for the listener to maintain on the selected protocol. Choose a number that is greater than 0 but no greater than the PRESPAWN_MAXIMUM value. The value should be about what you expect the average number of connections to be at any given time.
TIME OUT	Time in minutes that an inactive prespawned dedicated server process waits for the next connection. The value should be greater than 0. (A value of 0 will allow an inactive shadow process to continue indefinitely, thus wasting machine resources.) For greatest efficiency, Oracle Corporation recommends a short Time Out value. The time out is activated only after a prespawned dedicated server process has carried a connection and been disconnected. In other words,

prespawned dedicated server processes that are waiting for their first connection do not time out.

Here is an example of a SID_DESC section of LISTENER.ORA that includes information about prespawned dedicated server processes:

```
(SID_LIST =
    (SID_DESC =
      (GLOBAL_DBNAME = sales.acme.com
      (SID_NAME = DB1)
      (ORACLE_HOME = /usr/bin/oracle)
      (PRESPAWN_MAX = 99)
      (PRESPAWN_LIST=
           (PRESPAWN_DESC=
                 (PROTOCOL=TCP)
                 (POOL_SIZE=10)
                 (TIMEOUT = 2)
           )
      )
    )
)
```

LISTENER.ORA Control Parameters

The third section of the LISTENER.ORA file contains a list of parameters that control the behavior of the listener. The parameters and their defaults (if any) follow:

PASSWORDS_*listener_name*=(*password*[,...*password*])

This optional parameter allows one or more passwords. If this parameter is specified with one or more passwords, then the use of one of these passwords is required to perform certain DBA tasks against the listener using the Listener Control Utility. See “Using the Listener Control Utility” in Chapter 5 If this parameter is not included in the file, then anyone can access the Listener Control Utility to stop or alter the listener.

You may choose to have Network Manager encrypt the password in this file. If you choose to have the password encrypted, unauthorized people cannot see it in the LISTENER.ORA file and use it to manipulate the listener. However, if you want the password to be encrypted, you can enter only one password. The default is to encrypt the password.

If you do not care about password encryption, you can enter more than one password. If more than one

password is entered, all of them must be surrounded by parentheses. For example:

```
PASSWORDS_LISTENER=(super32, sly51)
```

Network Manager creates the parentheses automatically.

STARTUP_WAIT_TIME_*listener_name=number*

This parameter sets the number of seconds that the listener sleeps before responding to the first listener control status command. This feature assures that a listener with a slow protocol will have had time to start up before responding to a status request. Default is 0.

For example, in the case of SPX, if you use the Listener Control Utility to request a STATUS immediately after the START command, and if this parameter is set to 0, you will get an error stack indicating that the listener is not available. If this parameter is set to 2, however, the Listener Control Utility will wait briefly, and the STATUS command will return a message showing that the listener is available and listening. (For information about using the Listener Control Utility, see Chapter 5.)

CONNECT_TIMEOUT_*listener_name=number*

This parameter sets the number of seconds that the listener waits to get a valid SQL*Net version 2 connection request after a connection has been started. The listener drops the connection if the timeout is reached. Default is 10; if set to 0, it will wait forever.

TRACE_LEVEL_l*istener_name*=OFF | USER | ADMIN

This parameter indicates the level of detail the trace facility records for listener events. Choices are OFF, USER,or ADMIN. Default is OFF. USER provides a limited level of tracing; ADMIN provides a more detailed trace.

TRACE_DIRECTORY_*listener_name=path_to_trace_directory*

This parameter sets the directory where the trace file is placed. Default is operating system specific.

On UNIX, for example, it is $ORACLE_HOME/network/trace.

TRACE_FILE_*listener_name=trace_filename*

This parameter establishes the name of the file to which trace information is written. Default is *listener_name*.trc on most platforms.

LOG_DIRECTORY_*listener_name=path_to_log_directory*

This parameter indicates the directory in which to find the log file that is automatically generated for listener events. Default is operating system specific. On UNIX, for example, it is $ORACLE_HOME/network/log.

LOG_FILE_*listener_name=log_filename*

This parameter sets the name of the log file for the listener. Default is *listener_name*.log on most platforms.

Sample LISTENER.ORA File

In this example, each element is laid out on a separate line, so that it is easy to see the file's structure. This is the recommended format, and this is how the files created by Network Manager look. If you must edit a LISTENER.ORA file by hand, you do not have to put each element on a separate line. Be careful, though, to include all the appropriate parentheses, and to indent if you must continue an element onto the next line. Again, we strongly recommend that you use Network Manager. This example assumes the UNIX operating system.

```
LISTENER_mike.world=(ADDRESS_LIST=
          (ADDRESS=
                (PROTOCOL=IPC)
                (KEY=prod.world)
          )
          (ADDRESS=
                (PROTOCOL=IPC)
                (KEY=db1)
          )
          (ADDRESS=
                (PROTOCOL=tcp)
                (HOST=mike.world)
                (PORT=1521)
          )
     )
SID_LIST_LISTENER=
```

```
        (SID_DESC=
              GLOBAL_DBNAME=sales.acme.com)
              (SID_NAME=db1)
              (ORACLE_HOME=/usr/oracle7)
        )

#The following parameters have default values

PASSWORDS_LISTENER=
STARTUP_WAIT_TIME_LISTENER=0
CONNECT_TIMEOUT_LISTENER=10
TRACE_LEVEL_LISTENER=OFF
TRACE_DIRECTORY_LISTENER=/usr/prod/oracle7/network/trace
TRACE FILE_LISTENER=listener.trc
LOG_DIRECTORY_LISTENER=/usr/prod/oracle7/network/log
LOG_FILE_LISTENER=listener.log
```

OS Doc

Note that if the listener were on a different operating system, the default file and path names for tracing and logging might be different. See your Oracle operating system-specific documentation for further information.

Controlling the Network Listener

The listener is controlled by the Listener Control Utility, LSNRCTL. For information on how to use LSNRCTL to start and control the listener, refer to Chapter 5, ”Using SQL*Net”.

The Server as a Client

When the server does more than its traditional role of receiving connections and responding to queries, it requires the client configuration. Server initiated connections are technically identical to client connections. Specifically, when using database links to initiate connections to other servers, the server needs all of the same configuration information as a client. It must have access to the TNSNAMES.ORA file and a SQLNET.ORA file. If it might use an Interchange, it must have its own TNSNAV.ORA file. The server has the following requirements:

- The Oracle Server needs access to the connect descriptors for any other servers it will be contacting.
- The Oracle Server needs to define the communities of which it is a member.
- The server requires one or more preferred Connection Managers if it will be performing database links through an Interchange and if there is more than one Interchange in the network. A Connection Manager is part of an Interchange. The Connection Manager and the TNSNAV.ORA file are described in the *Oracle MultiProtocol Interchange Administrator’s Guide.*

Intchg

The following sections, which describe TNSNAMES.ORA and SQLNET.ORA for clients, are therefore applicable to servers as well. Oracle Network Manager automatically generates these files for every server.

Identifying Destinations: TNSNAMES.ORA

The TNSNAMES.ORA file is used by clients and distributed database servers to identify potential destinations, both servers and, optionally, Interchanges. (If Oracle Names is used in the network, the TNSNAMES.ORA files are not necessary; the Names Servers get the needed information from the network definition stored on a database. Similarly, if an Oracle Native Naming Adapter such as NIS or DCE's CDS is used in the network, then service names can be resolved by one or more native naming services.) Unless you are using Oracle Names or an Oracle Native Naming Adapter, Network Manager generates the TNSNAMES.ORA file. Each entry in the TNSNAMES.ORA file includes two elements:

- a service name
- a connect descriptor

These elements are described in the following sections.

Service Names

All connect descriptors are assigned *service names* in the TNSNAMES.ORA file. The user specifies the service name, a single word rather than the lengthier connect descriptor, to identify the service to which the user wants to connect. (These are comparable to the aliases used for connect_strings in SQL*Net version 1.) The contents of a TNSNAMES.ORA file consists of a series of service names mapped to TNS connect descriptors.

The service name for a database must be exactly the same as the global database name defined by the system administrator. SQL*Net limits the total length of a global database name to 64 characters. Of these, up to eight are the DB_NAME as defined by the database administrator, and the remainder show the service's place in the domain hierarchy (DB_DOMAIN). The name part of the service name can be longer than eight characters only if the DBA changes the name of the database with a RENAME GLOBAL_NAME parameter. The total global database name, or service name, must remain at or below 64 characters. See "Global Naming Issues" in Chapter 2 of *Oracle7 Server Distributed Systems, Volume I* for more information on creating a global database name.

Alternate service names, or aliases, may be assigned to a database service through the TNSNAMES.ORA file. The alternate service names may be any convenient, easy to remember names you choose. For example, if a database were used by two different divisions of a company, Human Resources and Finance, you might want to map two different service name aliases, HR and FINANCE, to the database. The TNSNAMES.ORA file would then have three separate entries: a service name that is the same as the global database name, and two aliases, mapped to the same connect descriptor.

The service name for an Interchange is the name of the Interchange or its Connection Manager component. Typically, the Interchange and the Connection Manager are referred to by the same name.

Connect Descriptors

Every service requires a connect descriptor. For a database, a connect descriptor describes the location of the network listener and the system identifier (SID) of the database to which to connect. Database connect descriptors commonly consist of two sections:

- the listener ADDRESS
- the database SID passed as application CONNECT_DATA

ADDRESS Section

The application address is the information required to reach the application within a given protocol environment. It includes the community in which the destination resides, the protocol it uses, and protocol–specific parameters. Oracle Network Manager automatically provides the correct protocol specific parameters for any protocol you use, but you must provide the appropriate values. For information about the values for the parameters for a given protocol, see the Oracle operating system-specific documentation for your platform.

OS Doc

> **Note:** If you specify a TCP/IP address prefixed with a ”0”, it is assumed to be an octal number, not a decimal number. For example, 39.223.72.44 is a decimal number, but 039.223.72.44 is an octal number.

CONNECT_DATA Section

SQL*Net uses the CONNECT_DATA keyword to denote the system identifier (SID) of the remote database. When SQL*Net on the server side receives the connection request, TNS passes the CONNECT_DATA contents to the listener, which identifies the desired database. For SQL*Net use, sample CONNECT_DATA might look like:

```
(CONNECT_DATA=
   (SID=V7PROD)
)
```

CONNECT_DATA is a protocol independent keyword indicating that application–specific data will be supplied at connect time, and SID specifies the Oracle System ID of the database server. You must specify the SID in the CONNECT DATA section of the connect descriptor.

With this release of SQL*Net, the CONNECT_DATA section must also include the global database name of the database. In most instances the global database name is the same as the service name. However, if you have a replicated database, you may have assigned a single service name that maps to more than one database name.

Interchange Addresses

A connect descriptor for an Interchange consists of only one section, an ADDRESS_LIST section. Within the ADDRESS_LIST section all the addresses of the Interchange are listed, including the required protocol specific keywords.

There is no CONNECT_DATA section in the connect descriptor of an Interchange.

Example of the Use of TNSNAMES.ORA

Figure A – 2 shows a simple network in which client applications access a database on the server NY_VAX. In this example, the SID of the database is DB1. Its service name and connect descriptor are found in the TNSNAMES.ORA entry that follows.

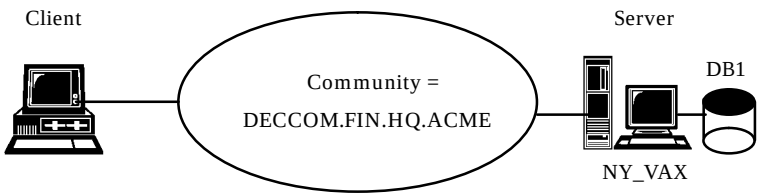

Figure A – 2 SQL*Net Connect Descriptor

The following TNSNAMES.ORA entry maps the service name NY_FIN to the connect descriptor:

```
NY_FIN.FIN.HQ.ACME = (DESCRIPTION=
                        (ADDRESS=
                          (COMMUNITY=DECCOM.FIN.HQ.ACME)
                          (PROTOCOL=DECNET)
                          (NODE=NY_VAX.FIN.HQ.ACME)
                          (OBJECT=LSNR)
                        )
```

```
    (CONNECT_DATA=
      (SID=DB1)
      (GLOBAL_NAME=NY_FIN.FIN.HQ.ACME)
    )
)
```

A user who wished to access the database would use NY_FIN to identify the appropriate connect descriptor. For example:

```
SQLPLUS SCOTT/TIGER@NY_FIN
```

This example assumes that NY_FIN is in the user's default domain. If it is not, the whole service name (NY_FIN.FIN.HQ.ACME) would need to be used.

Updating connect descriptors

Whenever you add a new database to the network, you must add a new service name and connect descriptor to the TNSNAMES.ORA file. Use Oracle Network Manager to update TNSNAMES.ORA.

System and User TNSNAMES.ORA Files

On most platforms, there can be two versions of the TNSNAMES.ORA file, one at the system level (all users) and an optional private one at the user level. If a private TNSNAMES.ORA file exists, its contents take precedence over the system–level file. That is, if both files have the same service name mapped to different connect descriptors, the connect descriptor in the user's local file will be used.

A local TNSNAMES.ORA file does not replace the system file, but exists in addition to it. For example, if a developer creates a database which is not generally accessible to other users, he might want to create a private TNSNAMES.ORA file with a simple service name mapped to its connect descriptor. By creating his own TNSNAMES.ORA file, the developer can have the convenience of using a service name without having to go through the system administrator.

OS Doc

Note: For more specific information about the name, location, and use of the TNSNAMES.ORA file on your operating system, refer to your Oracle operating system–specific manual.

Configuring Listener Load Balancing

To configure listener load balancing, the administrator must configure multiple listeners for each database. There can be multiple listeners on the same platform as the database, or, for multi–threaded servers, the listeners can be on different nodes. For multi–threaded servers, the administrator must add some parameters to the database parameter file. (On UNIX systems, this is INIT.ORA.)

Note: The administrator does not need to add or change any parameters in the database parameter file for dedicated servers.

Database Parameter File

For multiple listeners to be enabled for multi–threaded servers, the database initialization parameter file must include the following parameter:

```
MTS_MULTIPLE_LISTENERS=TRUE
```

> **Note:** Unless MTS_MULTIPLE_LISTENERS=TRUE is included, listener load balancing is not enabled.

If there is a single listener for each multi–threaded server, each address of the single listener is listed on a separate line, as follows:

```
MTS_LISTENER_ADDRESS= address
MTS_LISTENER_ADDRESS= address
[MTS_LISTENER_ADDRESS= address]
...
```

If multiple listeners are enabled, each listener must be on a separate line. If a listener has multiple addresses, each one is listed within an address list, as follows:

```
MTS_LISTENER_ADDRESS=(ADDRESS_LIST=(ADDRESS=(address)
      (ADDRESS=(address)))
MTS_LISTENER_ADDRESS=(ADDRESS_LIST=(ADDRESS=(address)
      (ADDRESS=(address)))
```

> **Note:** If you decide to change back so that each dispatcher registers with only one listener, you must remove the ADDRESS_LIST keyword and place alternative addresses for the same listener on separate lines:
>
> ```
> MTS_LISTENER_ADDRESS=address
> MTS_LISTENER_ADDRESS=address
> ```

TNSNAMES.ORA

For clients to be able to randomize connections between listeners, a new address format is needed in TNSNAMES.ORA, as follows:

```
service_name =
  (DESCRIPTION_LIST=
    (DESCRIPTION=(ADDRESS_LIST=(ADDRESS=address)
                  (ADDRESS=address))
                   (CONNECT_DATA=(SID=sid)
                        (GLOBAL_NAME=global_dbname)))
    (DESCRIPTION=(ADDRESS_LIST=(ADDRESS=address)
                         (ADDRESS=address))
                 (CONNECT_DATA=(SID=sid)
```

```
                                (GLOBAL_NAME=global_dbname)))
    [(CONNECT_DATA =(SID=sid)(GLOBAL_NAME=global_dbname))])
```

In this syntax, each description in the description list is for a different listener. If the connect data is the same for all the descriptions in a description list, it need be entered only once, in the last line.

The global name is a concatenation of the *db_name* and the *db_domain* from the INIT.ORA file.

> **Note:** The old format for TNSNAMES.ORA entries continues to work only for those databases that have only one listener.

TNSNAMES.ORA Example for a Single Database Instance with Two Listeners

```
emp=(DESCRIPTION_LIST=
     (DESCRIPTION=(ADDRESS_LIST=
       (ADDRESS=(COMMUNITY=DECCOM.WORLD)(PROTOCOL=DECNET)
                (NODE=NY_VAX)(OBJECT=LSNR))
       (ADDRESS=(COMMUNITY=TCPCOM.WORLD)(PROTOCOL=TCP)
                (HOST=NY_VAX)(PORT=1526))))
    (DESCRIPTION=(ADDRESS_LIST=
       (ADDRESS=(COMMUNITY=DECCOM.WORLD)(PROTOCOL=DECNET)
                (NODE=NY_SEQ)(OBJECT=LSNR))
       (ADDRESS=(COMMUNITY=TCPCOM.WORLD)(PROTOCOL=TCP)
                (HOST=NY_SEQ)(PORT=1526))))
   (CONNECT_DATA=(SID=db1)(GLOBAL_NAME=emp.world)))
```

In this example, there are two listeners for the database, on two different nodes, each listening on two different protocols. This example matches the example illustrated in Figure 2 – 4.

Many–to–Many Relationships

If equivalent databases use listener load balancing, a single service name may have descriptions with different SIDs and global database names. There may be several listeners, each listening for several database instances. In this case, there is Connect Data in every description, and no final Connect Data without an associated address.

If there are replicated databases that are equivalent, create a service name that maps to more than one SID and global database name. For example, the following portion of a TNSNAMES.ORA file is for a replicated server with two listeners. The example matches that shown in Figure 2 – 5.

TNSNAMES.ORA Example for Equivalent Databases

```
twinemp=(DESCRIPTION_LIST=
          (DESCRIPTION=(ADDRESS_LIST=
             (ADDRESS=(COMMUNITY=DECCOM.WORLD)
                      (PROTOCOL=DECNET)
                    (NODE=NY_VAX)(OBJECT=LSNR))
             (ADDRESS=(COMMUNITY=TCPCOM.WORLD)
                      (PROTOCOL=TCP)
                      (HOST=NY_VAX)(PORT=1526)))
          (CONNECT_DATA=(SID=db1)(GLOBAL_NAME=emp.world)))
          (DESCRIPTION=(ADDRESS_LIST=
             (ADDRESS=(COMMUNITY=DECCOM.WORLD)
                      (PROTOCOL=DECNET)(NODE=NY_SEQ)
                      (OBJECT=LSNR))
             (ADDRESS=(COMMUNITY=TCPCOM.WORLD)
                      (PROTOCOL=TCP)(HOST=NY_SEQ)
                      (PORT=1526)))
          (CONNECT_DATA=(SID=db2)(GLOBAL_NAME=hr.world))))
```

Configuring Clients: SQLNET.ORA

The SQLNET.ORA file is created for all clients and nodes on the network. It contains five types of information:

- the amount of time between probes sent to determine whether a client–server connection is still alive (dead connection detection)
- optional tracing and logging parameters
- default domains
- names resolution services
- client parameters for use with Oracle Names
- other optional parameters

These parameters are described in the following sections.

Dead Connection Detection

The optional parameter SQLNET.EXPIRE_TIME determines how often SQL*Net sends a probe to verify that a client–server connection is still active. If a client is abnormally terminated, a connection may be left open indefinitely unless identified and closed by the system. If this parameter is specified, SQL*Net sends a probe periodically to determine whether there is an invalid connection that should be terminated. If it

finds a dead connection, or a connection no longer in use, it returns an error, causing the server process to exit.

Specify this parameter in the Connection Expire Time field of the Client Profile property sheet of Network Manager. Enter the time, in minutes, between probes for a dead connection. The range of possible values is from one to a very large number. However, a value of approximately 10 is recommended. If no value is entered in this field, the broken connections may be maintained indefinitely.

> **Note:** The time set in this parameter is not necessarily the amount of time a dead connection will remain. This parameter sets the time between probes for dead connections. Depending on the underlying protocol, it may be somewhat longer before a dead process is shut down.

Dead connection detection has some costs associated with it.

- Additional network traffic is generated to probe for dead connections. A probe packet is very small, but one is sent on each connection at the interval specified in the SQLNET.EXPIRE_TIME parameter in the SQLNET.ORA file.
- When dead connection detection is enabled, the Oracle Server may need to do additional processing to distinguish the connection probing event from other events that occur, depending on which operating system is in use. For example, additional processing is required on UNIX. As a result, there could be some small amount of degradation of performance in the server. The best way to determine whether you will be affected is to test the performance of your application with and without the dead connection detection feature enabled.
- For some protocols, the generic SQL*Net dead connection detection feature is no better than the native mechanism available in the underlying transport protocol. In that case, it is not necessary to enable it.

In short, you should evaluate carefully whether you would benefit from enabling the dead connection detection feature. It should only be turned on if necessary.

Optional Logging and Tracing Parameters

If you select any optional logging and tracing parameters in the Client Profile property sheet of Network Manager, the following parameters appear in the SQLNET.ORA file:

- LOG_FILE_CLIENT
- LOG_DIRECTORY_CLIENT

- TRACE_LEVEL_CLIENT
- TRACE_FILE_CLIENT
- TRACE_DIRECTORY_CLIENT

Note: Comparable parameters may be created for servers (for example, LOG_FILE_SERVER, TRACE_DIRECTORY_SERVER). You must add any logging and tracing parameters for servers to the SQLNET.ORA file by hand; they are not generated by Network Manager or by the SQLNET.ORA Editor.

You can add the following optional tracing parameters for the TNSPING utility to SQLNET.ORA using the SQLNET.ORA Editor. (They produce messages similar to the SQL*Net trace parameters mentioned above.)

- TNSPING.TRACE_LEVEL
- TNSPING.TRACE_DIRECTORY

Diagnostics

For more information about the logging and tracing parameters in SQLNET.ORA, see Chapters 2and 3 in the *Oracle Network Products Troubleshooting Guide*.

Default Domains

Whether or not you are using Oracle Names, the SQLNET.ORA file includes a parameter that shows the default domain. This parameter is normally set through Network Manager, but can be modified using the SQLNET.ORA Editor.

Name Resolution Services

The order in which different names resolution services are attempted is determined by the NAMES.DIRECTORY_PATH parameter. By default, TNSNAMES.ORA file is the first method listed, with Oracle Names next. If you have installed a native naming adapter, it would appear on the list also. You can change the order in which the names services appear, and therefore are used by the client, by changing them in the Client Profile using Network Manager. By default, Network Manager generates the following parameter and value in the SQLNET.ORA file:

```
NAMES.DIRECTORY_PATH = (TNSNAMES, ONAMES)
```

Oracle Names Parameters

ONames

If you are using Oracle Names, another parameter, NAMES.PREFERRED_SERVERS, is required. This parameter includes one or more addresses of the Names Servers the client prefers to use. Several optional Oracle Names tracing parameters may also appear; they are described in Appendix B of the *Oracle Names Administrator's Guide*. These parameters are created using the Oracle Network Manager.

Additional SQLNET.ORA Parameters

The SQLNET.ORA file is used primarily for specifying the Dead Connection Detection parameter, tracing parameters, and default

domain information. However, there are additional optional parameters which provide other functions you may find useful. The following parameters must be edited manually in the SQLNET.ORA file; they are not affected by Network Manager.

Turning Off IPCs

If for some reason you do not want IPC addresses to be sought automatically on some nodes in your network, you should add the following parameter to the SQLNET.ORA files for those nodes:

AUTOMATIC_IPC=OFF

Without this parameter, the default is for a connection to look for an IPC address.

Using a Dedicated Server

Generally when the listener receives a connection request, it hands off the request to an existing process. That is, it makes use of the multi–threaded server. If you prefer that the listener spawn a dedicated server task or process for connections to this database, add the following line to the SQLNET.ORA file for the listener node:

USE_DEDICATED_SERVER=ON

The default is OFF.

> **Note:** For further information about dedicated server processes and the multi–threaded server, see Chapter 2 in the *Oracle7 Server Distributed Systems, Volume I*.

It is important that any manually–edited entries be made in SQLNET.ORA after the file is distributed to the destination. Because this file is generated to be common to all clients with the same client profile, changing it for specific nodes or a specific operating system's file structure before moving it to other nodes may have unintended results.

> **Note:** Also, you can configure a service name alias to use a dedicated server process (instead of the multi–threaded server) in Network Manager. After generating configuration files, then add the USE_DEDICATED_SERVER=ON to the SQLNET.ORA file for the listener node.

Disabling Out of Band Breaks

If you want to disable out-of-band breaks, set the DISABLE_OOB=ON parameter in SQLNET.ORA on clients (and servers that will act as clients) to ON. The default is OFF. If you are using an Interchange, out-of-band breaks are not implemented, regardless of how this parameter is set.

```
DISABLE_OOB=OFF|ON
```

Currently, you need to add this parameter manually to SQLNET.ORA; you cannot configure it through Network Manager.

Sample SQLNET.ORA Files

This sample SQLNET.ORA file is for a client in a network that does not include Oracle Names and that accepts the default trace parameters. No value is specified for SQLNET.EXPIRE_TIME, which means that dead connection detection is not enabled.

```
TRACE_LEVEL_CLIENT = OFF
sqlnet.expire_time =
names.default_domain = world
name.default_zone = world
```

The NAMES.DEFAULT_DOMAIN parameter is created whether or not Oracle Names is part of the network. Oracle Network Manager puts a comparable parameter, NAME.DEFAULT_ZONE, into the file for backward compatibility with an earlier version of SQL*Net.

The following SQLNET.ORA file is for a client in a network that includes Oracle Names. A value has been included for SQLNET.EXPIRE_TIME.

```
TRACE_LEVEL_CLIENT = OFF
sqlnet.expire_time = 10
names.default_domain = world
name.default_zone = world
names.preferred_servers =
      (ADDRESS_LIST =
        (ADDRESS =
          (COMMUNITY = tcpcom)
          (PROTOCOL = TCP)
          (Host = vance.world)
          (Port = 1522)
        )
      )
name.preferred_servers =
      (ADDRESS_LIST =
        (ADDRESS =
          (COMMUNITY = tcpcom)
          (PROTOCOL = TCP)
          (Host = vance.world)
          (Port = 1522)
        )
      )
```

This file includes a parameter that shows the client's preferred Names Server. As in the previous sample, there are two different versions of this parameter to provide backward compatibility with an earlier version of

the product. Oracle Network Manager creates these parameters automatically.

Creating Special Address Parameters: PROTOCOL.ORA

PROTOCOL.ORA defines node specific and protocol specific addressing information for certain protocols. It also includes Validnode verification on those protocols that support it.

Protocol–Specific Address Parameters

The following protocols require parameters to be listed in PROTOCOL.ORA:

- APPC / LU6.2
- ASYNC
- X.25
- OSI4

This file contains node–specific non–global address parameters and other protocol specific configuration parameters. Protocols that require address information in PROTOCOL.ORA typically have LOCAL_LOOKUP=*alias* as one of their address parameters in TNSNAMES.ORA, LISTENER.ORA, or TNSNET.ORA. The LOCAL_LOOKUP parameter points to a non–global address in a PROTOCOL.ORA file. Oracle Network Manager does not create PROTOCOL.ORA files; they must be created by hand.

For example, consider the partial TNSNAMES.ORA, LISTENER.ORA, and PROTOCOL.ORA files for LU6.2 shown in Figure A – 3.

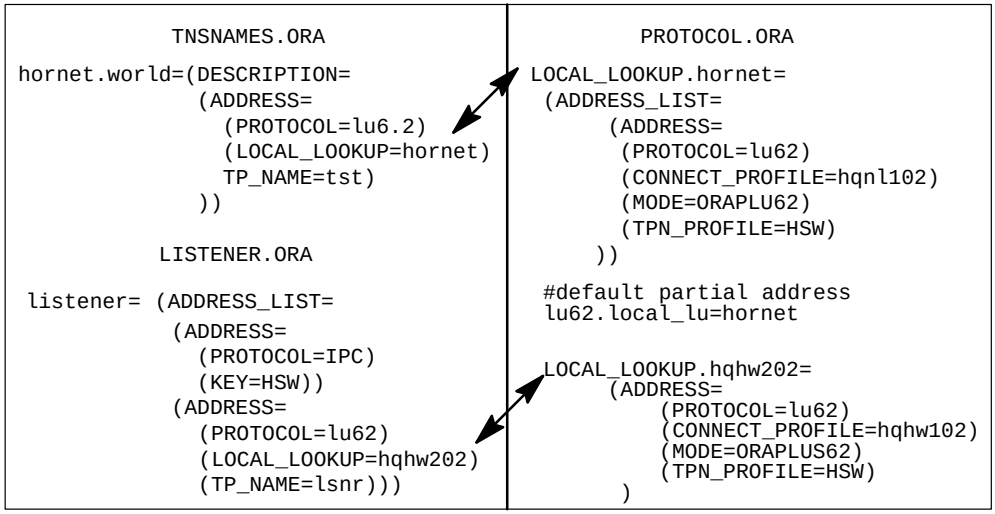

Figure A – 3 LOCAL_LOOKUP and PROTOCOL.ORA

The global information address information for the server HORNET.WORLD is contained in the TNSNAMES.ORA and LISTENER.ORA files. This information can be used by any client in the network. The PROTOCOL.ORA entry contains additional address parameters needed for a specific node to reach HORNET.WORLD.

OS Doc

See the Oracle operating system specific documentation for your platform for further information.

Validnode Verification

The objective of Validnode verification is to restrict connection access of network clients to those with enabling host privilege. The access list is in the PROTOCOL.ORA file. The list is dynamic and used by the Validnode component to decide on granting access to incoming connection requests.

To activate Validnode checking, the following parameter must be entered in PROTOCOL.ORA:

protocol.validnode_checking = yes

For example, for the TCP/IP protocol, the parameter would be:

tcp.validnode_checking = yes

The default is for Validnode checking to be off.

> **Note:** The DECnet protocol does not currently support Validnode checking.

The access list for Validnode checking can take two forms: an INVITED_NODES list and an EXCLUDED_NODES list. The two lists are mutually exclusive. If both are present, the INVITED_NODES list takes precedence over the EXCLUDED_NODES list.

Sample Access List

The following is a sample access list in PROTOCOL.ORA for the TCP/IP protocol:

```
tcp.invited_nodes = (drummer.us.com,
                     139.185.5.73
                     139.185.5.111)
```

This list grants access to the three nodes listed, and excludes all others.

An alternative way of limiting connection requests is to exclude specific nodes. The following list provides access to all nodes except those listed:

```
tcp.excluded_nodes = (drummer.us.com,
                      139.185.5.73
                      139.185.5.111)
```

OS Doc

Note: Not all protocols and operating systems support Validnode verification. See the operating system–specific documentation for your platform for further details.

Enabling SNMP Support: SNMP.ORA

NetMan

To enable SNMP support, you must create an SNMP.ORA file for every node containing a service. You do this by using the SNMP pages for the Oracle7 Server, listener, Oracle Names, and Interchange in Network Manager, and then generating configuration files. (See Chapter 5 in the *Oracle Network Manager Administrator's Guide* for information on these pages.)

If a node has more than one service, the SNMP.ORA file includes information for all of them. For example, the SNMP.ORA file for a particular node might contain information so that an Interchange and a Names Server can respond to SNMP queries from an SNMP–based network management system. Or, an SNMP.ORA file on a particular node might include parameters for a listener and a database only.

The sample SNMP.ORA file in this section enables SNMP support for five services including a listener, two databases (HR.WORLD and SALES.WORLD), an Interchange and a Names Server. These five services are configured to respond to queries from an SNMP–based network management system.

Sample SNMP.ORA

```
################
# Filename......: snmp.ora
# Node..........: heather.world
# Date..........: 11-MAR-95 14:23:45
################
SNMP.VISIBLESERVICES
  =(INTCHG_1,NameServer,hr,sales,LISTENER)
SNMP.INDEX.LISTENER = 20
SNMP.INDEX.hr = 10
SNMP.INDEX.INTCHG_1 = 30
SNMP.INDEX.NameServer = 40
SNMP.INDEX.sales = 50
SNMP.CONTACT.LISTENER = "jadmin, 415 528-6293"
SNMP.SID.hr = hr
SNMP.CONTACT.hr = "jadmin, 415 528-6293"
SNMP.CONTACT.INTCHG_1 = "jadmin, 415 528-6293"
SNMP.CONTACT.NameServer = "jadmin, 415 528-6293"
SNMP.SID.sales = sales
SNMP.CONTACT.sales = "jadmin, 415 528-6293"
```

SNMP.ORA Control Parameters

The SNMP.ORA file contains parameters that allow an Oracle7 Server, listener, MultiProtocol Interchange, or Oracle Names Server to be queried by any SNMP–based network management system. The parameters and their defaults (if any) follow:

SNMP.VISIBLESERVICES=(*svc–name–x, svc–name–y, svc–name–z...)*

This parameter specifies a list of SNMP–managed services. For each database or Oracle network service for which SNMP support is enabled, the name of that service must in the SNMP.VISIBLESERVICES list. The other SNMP.ORA parameters refer to these service names.

You must specify service names for database names in the SNMP.VISIBLESERVICES list, for example, HR or SALES. The service name can be an alias to a SQL*Net V1 connect string or the SQL*Net version 2 service name.

SNMP.INDEX.*SERVICE_NAME=unique–integer*

Each service listed in the SNMP.VISIBLESERVICES line must have an index entry of an integer 1 or greater.

Note: The integer must be unique on the host.

The integer is used as the value of applIndex, which indexes the appl`Table` of the Network Services MIB, and almost all of the tables in the RDBMS MIB and private MIB.

Attention: The selected integer must not conflict with those assigned to other applications implementing the Network Services MIB on this host such as other databases, mail, or directory servers.

SNMP.SID.*SERVICE_NAME=SID*

This parameter specifies the ORACLE_SID associated with the database. ORACLE_SID is used by the database subagent as the value of applName for this database.

The subagent uses the SID in the following ways:

- Returns the SID as rdbmsSrvName
- Assumes that services with SIDs are databases
- Assumes that services without SIDs are other types of services

Note: If you do not include a SID for a database, the Subagent assumes that it is a non–database service, and will not connect to it or register any MIB variables for it.

SNMP.CONNECT.*SERVICE_NAME=username*

This parameter overrides the default username created for the database Subagent.

Note: This optional parameter is only for databases.

SNMP.CONNECT.*SERVICE_NAME.*PASSWORD=*password*

This parameter overrides the default username created for the database Subagent.

SNMP.CONTACT.*SERVICE_NAME=”Contact Name, Phone Number, etc.*

Note: This parameter is optional; however, a name and phone number are recommended. Network Manager automatically creates the quotation marks (”) around the value for this parameter.

This parameter can be used for any SNMP–managed service and answers queries for the following:

rdbmsDbContact

rdbmsSrvContact

oraNamesTnsContact

oraInterchgContactInfo

oraListnerContact

If this parameter is not used for SNMP service, then either noSuchName is returned for SNMPv1 or noSuchIndex for SNMPv2.

Note: You cannot configure the following seven parameters from Network Manager. After you have generated configuration files with Network Manager, you must add them manually using a file editor of your choice.

DBSNMP.POLLTIME=*integer*

The integer signifies the interval (in seconds) to be used as a polling interval for the status of each SNMP–managed database. Default is 30 seconds.

The Subagent tries every 30 seconds to connect to SNMP–managed databases that are down.

DBSNMP.TRACE_LEVEL=OFF | USER | ADMIN

This parameter indicates the level of detail the trace facility records for database subagent events. Choices are OFF, USER, or ADMIN. Default is OFF. Selecting USER provides a limited level of tracing; ADMIN provides a more detailed trace.

DBSNMP.TRACE_DIRECTORY=*path_to_trace_directory*

This parameter sets the directory where the trace file is placed. Default is operating system specific. On UNIX, for example, it is $ORACLE_HOME/network/trace.

DBSNMP.TRACE_FILE=*trace_filename*

This parameter establishes the name of the file to which trace information is written. Default is DBSNMP.TRC on most platforms.

DBSNMP.TRACE_UNIQUE=ON | OFF

If the value is set to ON, a process identifier is appended to the name of each trace file generated so that several can coexist. The default is OFF.

DBSNMP.LOG_DIRECTORY=*path_to_log_directory*

This parameter indicates the directory in which to find the log file that is automatically generated for SNMP events. Default is operating system specific. On UNIX, for example, it is $ORACLE_HOME/network/log.

DBSNMP.LOG_FILE=*log_filename*

This parameter sets the name of the log file for Oracle SNMP support. Default is DBSNMP.LOG on most platforms.

Increasing the Queue Size for the Listeners

You can increase the listening queue, or backlog, for a listening process (that is, for TNSLSNR, INTLSNR, NAMES) to dynamically handle larger numbers of concurrent connection requests by adding a (QUEUESIZE=*n*) parameter to the end of any listening address in LISTENER.ORA, TNSNET.ORA, or NAMES.ORA. The parameter must be added as the last parameter in the network address (see sample files).

Note: Currently, you need to add this parameter manually to LISTENER.ORA, TNSNET.ORA, and NAMES.ORA—you cannot set it through Network Manager.

Note: Currently, you can configure the QUEUESIZE parameter for listeners on DECnet and TCP/IP protocols only. Also, if the QUEUESIZE parameter is not present in the appropriate configuration file (TNSNET.ORA, LISTENER.ORA, or NAMES.ORA), TCP/IP defaults to 17.

Increasing the Queue Size for the Listener

This section describes how to increase the queue size for the network listener. Add this parameter as the last parameter in the network address (see sample file). Currently, you need to add this parameter manually to LISTENER.ORA—you cannot set it through Network Manager. For information on other configuration parameters in LISTENER.ORA refer to ”Configuring the Network Listener: LISTENER.ORA” in this chapter.

LISTENER.ORA

```
################
# Filename......: listener.ora
# Node..........: mike.world
# Date..........: 11-OCT-94 14:23:45
################

 LISTENER_mike.world=(ADDRESS_LIST=
                              (ADDRESS=
                                 (PROTOCOL=IPC)
                                 (KEY=prod.world)
                              )
                              (ADDRESS=
                                 (PROTOCOL=IPC)
                                 (KEY=db1)
                              )
                              (ADDRESS=
                                 (PROTOCOL=tcp)
                                 (HOST=mike.world)
                                 (PORT=1521)
                                 (QUEUESIZE=20)
                              )
                    )
```

Increasing the Queue Size for the Interchange Listener

This section describes how to increase the queue size for the Interchange listener. Add this parameter as the last parameter in the network address (see sample file). Currently, you need to add this parameter manually to TNSNET.ORA—you cannot set it through Network Manager. For information on other configuration parameters in TNSNET.ORA, see the *Oracle MultiProtocol Interchange Administrator's Guide.*

TNSNET.ORA

```
################
# Filename......: tnsnet.ora
# Node..........: heather.world
# Date..........: 11-OCT-94 14:23:45
################
COMMUNITY_COST =
  (COMMUNITY_COST_LIST =
    (COMMUNITY_DESC =
      (COMMUNITY = spxcom.world)
      (COST = 10)
    )
    (COMMUNITY_DESC =
      (COMMUNITY = tcpcom.world)
      (COST = 10)
    )
  )
INTCHG_1.world =
    (ADDRESS_LIST =
        (ADDRESS =
          (COMMUNITY = spxcom.world)
          (PROTOCOL = SPX)
          (Service = heather_INT_1)
        )
        (ADDRESS =
          (COMMUNITY = tcpcom.world)
          (PROTOCOL = TCP)
          (Host = heather)
          (Port = 1526)
          (QUEUESIZE=20)
        )
    )
```

Increasing the Queue Size for the Oracle Names Server

This section describes how to increase the queue size for the Names Server. The parameter should be added as the last parameter in the network address (see sample file). Currently, you need to add this parameter manually to NAMES.ORA; you cannot set it through Network Manager. For information on other configuration parameters in NAMES.ORA, see Chapter 6 in the *Oracle Names Administrator's Guide.*

ONames

NAMES.ORA

```
names.server_name = NameServer.us.oracle.com
names.admin_region = (REGION=
                (NAME= LOCAL_REGION.world)
                (TYPE= ROSDB)
                (USERID= NETADMIN)
                (PASSWORD= netadmin)
                DESCRIPTION=
                 (ADDRESS_LIST=
                  (ADDRESS=
                   (COMMUNITY=TCPCOM.us.oracle.com)
                   (PROTOCOL=TCP)
                   (Host=deer.us.oracle.com)
                   (Port=1521)
                   (QUEUESIZE=20)
                  )
                 )
                 (CONNECT_DATA=(SID=ds)
                 )
                )
                (DOCNAME=deer)
                (VERSION= 34611200)
                (RETRY = 600))
names.config_checkpoint_file= cfg00406
names.trace_level= OFF
names.trace_unique= FALSE
```

APPENDIX

B

Syntax Rules for Configuration Files

This appendix describes the syntax rules for configuration files. Following these rules is extremely important; even small errors in formatting make a configuration file unusable. For this reason, Oracle Corporation strongly recommends the use of the Oracle Network Manager whenever possible. Not only is using the tool faster, easier, and more accurate than creating the files by hand, but Oracle supports only files that can be reproduced using the tool (except for SQLNET.ORA and PROTOCOL.ORA files, which contain parameters that can be included only through manual editing).

This appendix also includes information about the acceptable characters to use in various components of the files.

The contents of the files are described in detail in Appendix A.

Keyword–Value Pair Syntax

Most of the configuration files required in a SQL*Net version 2 network are made up in part of parameters followed by keyword–value pairs. Keyword–value pairs following parameters are surrounded by parentheses:

```
parameter = (keyword=value)
```

Some keywords have other keyword–value pairs as their values; in these cases, the group of keyword–value pairs that logically belong together are surrounded by parentheses:

```
(keyword=
    (keyword=value)
    (keyword=value)
)
```

For example, the address portion of a TNSNAMES.ORA file might include the following lines:

```
(ADDRESS=
    (COMMUNITY=tcpcom.hq.fin.acme)
    (PROTOCOL=tcp)
    (HOST=max)
    (PORT=1521)
)
```

Oracle Corporation recommends that you set up the files so that indentation reflects what keyword is the ”parent” or ”owner” of other keyword–value pairs. This format is not required, but it does make the files much easier to read and understand. This is the format that Network Manager generates.

Even if you do not choose to indent your files in this way, you must indent a wrapped line by at least one space, or it will be misread as a new parameter. The following layout is acceptable:

```
(ADDRESS=(COMMUNITY=tcpcom.world)(PROTOCOL=tcp)
    (HOST=max.world)(PORT=1521))
```

The following layout is *not* acceptable:

```
(ADDRESS=(COMMUNITY=tcpcom.world)(PROTOCOL=tcp)
(HOST=max.world)(PORT=1521))
```

Further Syntax Rules for TNS Configuration Files

The following rules apply to the syntax of configuration files:

- Any keyword in a configuration file that should be recognized as beginning a parameter that includes one or more keyword–value pairs must be in the far left column of a line. If it is indented by one or more spaces, it is interpreted as a continuation of the previous line.
- All characters must belong to the *network character set* (see the next section).
- Keywords are not case sensitive. Values may be case sensitive, depending on the operating system and protocol.
- Spaces around the "=" sign are optional in keyword–value pairs.
- There is an implied hierarchy of keywords. That is, some keywords are always followed by other keywords. At any level of the hierarchy keywords can be listed in any order. For example, the following entries are equally valid:

```
(ADDRESS =
       (COMMUNITY = TCPCOM.WORLD)
       (PROTOCOL = TCP)
       (HOST = MARTHA.WORLD)
       (PORT = 1521)
)

(ADDRESS =
       (COMMUNITY = TCPCOM.WORLD)
       (PROTOCOL = TCP)
       (PORT = 1521)
       (HOST = MARTHA.WORLD)
)
```

- Keywords cannot contain spaces. Values must not contain spaces unless enclosed within double quotes ("). For example, the following is a valid value:

```
(COMMUNITY = "West Central")
```

- The maximum length of a connect descriptor is 512 bytes.
- Comments can be included using the pound sign # at the beginning of a line. Anything following the sign to the end of the line is considered a comment.

Network Character Set

The network character set consists of the following characters. Values given for keywords must be made up only of characters in this set. Connect descriptors must be made up of single–byte characters.

```
A–Z, a–z
0–9
( ) < > / \
, . : ; ’ ” = – _
$ + * # & ! % ? @
```

Within this character set, the following symbols are reserved:

```
(  ) = \ ” ’ #
```

Reserved symbols should be used only as delimiters, not as part of a keyword or a value unless the keyword or value is quoted. Either single or double quotes can be used to enclose a value containing reserved symbols. To include a quote within a value that is surrounded by quotes, use different quote types. The backslash (\) is used as an escape character.

A specific example of the use of reserved symbols is in a numeric DECnet object within a DECnet address. As defined by DECnet, an OBJECT can be a name such as ABC or a value such as #123. These would be entered in the form:

```
(OBJECT=ABC)
```

or

```
(OBJECT=\#123)
```

The numeric DECnet object requires a symbol that is reserved by TNS. Because # is a reserved symbol, the character must be preceded by a backslash. See the Oracle Protocol Adapter information for your platform for further details on DECnet.

OS Doc

The following characters can be used within the structure of a connect descriptor, but cannot be part of a keyword or value:

```
<space> <tab> <CR> <newline>
```

Service Name Character Set

The listener name, service name, and Interchange names used in TNS configuration files are limited to the following character set:

```
[a...z] [A...Z] [0...9] _
```

The first character in a service name must be an alphabetic character. The number of characters allowed is platform specific; in general, however, up to 64 characters is acceptable. A database service name must match the global database name defined by the DBA. It consists of a database name (originally limited to eight characters), and the database domains. Service names and global database names are not case sensitive.

> **Note:** Network Manager uses periods (.) when it appends domains to the names you enter, but it does not accept periods in the name you provide.

Creating the LISTENER.ORA File

The LISTENER.ORA file describes the location of the listener, the services it listens for, and various control parameters. The contents of this file are described fully in Appendix A. This section describes the format of the file.

Placement of the Listener Name

The listener name must begin in the left–most column of a line, with no empty spaces preceding it.

Defining the Listener Addresses

The first section of LISTENER.ORA includes listener addresses. Network listener addresses are entered in the LISTENER.ORA file in the following form:

```
listener_name=[(ADDRESS_LIST=]
          (ADDRESS=
                [(COMMUNITY=community name)]
                (PROTOCOL=protocol name)
                (protocol specific information))
          [(ADDRESS=
                [(COMMUNITY=community name)]
                (PROTOCOL=protocol name)
                (protocol specific information)))]
```

If the listener is listening on only one protocol, the ADDRESS_LIST keyword is not required; a single ADDRESS entry is sufficient. For a single community the form is:

```
listener_name=(ADDRESS=
          [(COMMUNITY=community name)]
          (PROTOCOL=protocol name)
          (protocol specific information))
```

The listener is able to listen on more than one protocol. For example, if the host machine is running both TCP/IP and DECnet (that is, it is in both a TCP/IP and a DECnet community), the listener on that machine has an address that includes both protocols.

IPC Addresses

In addition to listening on protocols, the listener listens on at least two IPC addresses that must be defined. In one, the key value is equal to the service name. This IPC address is used for connections from other applications on the same node. In the other IPC address, the key value is equal to the database system identifier (SID), which is described in the next section. This IPC address is used by the database dispatcher to identify the listener.

> **Note:** If the service name is the same as the SID, only one IPC address is needed, and only one is generated by Network Manager.

Sample Listener Addresses

```
LISTENER=(ADDRESS_LIST=
          (ADDRESS=
                (PROTOCOL=IPC)
                (KEY=ny_finance.hq.fin.acme)
          )
          (ADDRESS=
                (PROTOCOL=IPC)
                (KEY=db3)
          )
          (ADDRESS=
                (COMMUNITY=tcpcomm)
                (PROTOCOL=TCP)
                (HOST=flash.hq.fin.acme)
                (PORT=1521)
          )
          (ADDRESS=
                (COMMUNITY=deccomm.hq.fin.acme)
                (PROTOCOL=DECNET)
                (NODE=flash.hq.fin.acme)
                (OBJECT=LSNR)
          )
     )
```

If you are creating the LISTENER.ORA file manually, and if there is only one community in your network, then the COMMUNITY keyword is optional for the listener, since listening is always performed on the local machine. You do, however, have the option to include it for consistency with your other connect descriptors. If you are using Network Manager you must provide the COMMUNITY keyword, but it is not included in the LISTENER.ORA file that is generated.

Describing the Databases on the Listener

The second section of the LISTENER.ORA file describes the system identifiers (SIDs) of the databases served by the listener. It is made up of keyword–value pairs as follows:

```
SID_LIST_listener_name=[(SID_LIST=]
                (SID_DESC=
                   (GLOBAL_DBNAME=global_database_name)
                   (SID_NAME=SID)
                   (operating_system_specific_
                     string=db_location)
                )
                [(SID_DESC=
                   (GLOBAL_DBNAME=global_database_name)
                   (SID_NAME=SID)
                   (operating_system_specific_
                    string=db_location)
                )]
              [)]
```

The SID_LIST keyword is required only if there is more than one database instance installed on the machine. In this format, *SID* is the Oracle system ID of the database server. In the next keyword–value pair, the keyword is operating system specific: it is indicated here as the variable *operating_system_specific_string*. Its value, indicated here as *db_location*, is the specific location of the database executables.

LISTENER.ORA Control Parameters

The third section of the LISTENER.ORA file contains a list of parameters that control the behavior of the listener. The parameters are entered in the following format:

```
PARAMETER=value
```

Do not allow any empty spaces on the line before the parameter name. Parentheses do not need to surround the parameter–value pairs.

Creating the TNSNAMES.ORA File

The TNSNAMES.ORA file contains the names and addresses of all the services on the network. The service names of databases are mapped to connect descriptors that describe their location on the network. Similarly, the service names of Interchanges are listed with their addresses on the network. The contents of this file are described fully in Appendix A. This section describes the format of the file.

> **Note:** TNSNAMES.ORA file is not needed if the network includes Oracle Names.

Placement of the Service Names

The service name must begin in the first space of a line to be parsed correctly. If you are editing the TNSNAMES.ORA file manually, be sure there are no spaces preceding the service name.

Connect Descriptors

Every database requires a connect descriptor. Database connect descriptors commonly consist of two sections:

- the listener ADDRESS
- the CONNECT_DATA, which includes the database SID and, if you are using listener load balancing, the global database name.

ADDRESS Section

The application address is the information required to reach the application within a given protocol environment. A sample valid ADDRESS is:

```
(ADDRESS=
    (COMMUNITY=COM1.HQ.FIN.ACME)
    (PROTOCOL=TCP)
    (HOST=LONDON_VAX.HQ.FIN.ACME)
    (PORT=1526)
)
```

An address is a set of keyword–value pairs related to a specific application. The keyword ADDRESS is shown at a different indentation, to show that it is the parent or owner of the other keywords. Each set of keyword–value pairs is surrounded by parentheses as shown. In this example, the protocol is identified as TCP/IP. TCP/IP requires two specific parameters: the name of the host computer as defined in TCP/IP, and the PORT number allocated to the network listener. (This may vary depending on what Oracle Network Manager generates. For example, for TCP, the port 1526 might be generated. In SQL*Net version 1, the port was usually 1525.)

OS Doc

For the parameters specific to other protocols, see the Oracle operating system-specific documentation for your platform.

CONNECT DATA Section

SQL*Net uses the CONNECT_DATA keyword to denote the system identifier (SID) and global database name of the remote database. When SQL*Net on the server side receives the connection request, TNS passes the CONNECT_DATA contents to the listener, which identifies the desired database. For SQL*Net use, sample CONNECT_DATA might look like:

```
(CONNECT_DATA=
   (SID=V7PROD)
   (global_name=prod.acme.com
)
```

CONNECT_DATA is a protocol-independent keyword indicating that application–specific data will be supplied at connect time. You must specify the SID in the CONNECT DATA section of the connect descriptor. In SQL*net 2.3, you are also expected to provide the global database name, which is made up of the database name and database domain from the database parameter file.

Interchange Addresses

A connect descriptor for an Interchange consists of only one section: an ADDRESS_LIST section. Within the ADDRESS_LIST section all the addresses of the Interchange are listed, including the required protocol specific keywords.

There is no CONNECT_DATA section in the connect descriptor of an Interchange.

> **Note:** Interchange names and addresses are optional in the TNSNAMES.ORA file. However, they are needed if you wish to access Interchanges from clients using the Interchange Control Utility. Oracle Network Manager automatically includes them in the TNSNAMES.ORA files it generates.

Creating the SQLNET.ORA File for Clients and Servers

SQLNET.ORA is made up primarily of simple keyword–value pairs. Each keyword–value pair starts at the left–most column of a line, with no empty spaces preceding it. If the keyword–value pair consists of a single word or a concatenation of words on either side of the equal sign, no parentheses are needed. Spaces around the equal sign are optional.

Similarly, if you do not want to use IPC addresses and you do not want to use the multi–threaded server, you would add the following keyword–value pairs to the file:

```
AUTOMATIC_IPC = OFF
USE_DEDICATED_SERVER = ON
```

This syntax applies to all the parameters you might need to add manually to the SQLNET.ORA file.

If the network includes Oracle Names, the SQLNET.ORA file also contains addresses of the Names Servers preferred by the client or server. These addresses are entered by Network Manager and should not be edited. They have the same syntax as addresses in the LISTENER.ORA file described earlier. For example, a server in a TCP/IP community might have a preferred Names Server listed as follows:

```
names.preferred_servers =
   (ADDRESS_LIST =
     (ADDRESS =
          (PROTOCOL = TCP)
          (Host = iris.west.acme.com)
          (Port = 1522)
     )
   )
name.preferred_servers =
   (ADDRESS_LIST =
     (ADDRESS =
          (PROTOCOL = TCP)
          (Host = iris.west.acme.com)
          (Port = 1522)
     )
   )
```

The address is listed twice with slightly different keywords to provide backward compatibility to an earlier version.

> **Note:** Do not edit the addresses of preferred Names Servers manually. Create and modify them using the Oracle Network Manager.

Creating the PROTOCOL.ORA File

OS Doc

The PROTOCOL.ORA file follows the same rules of syntax as the SQLNET.ORA file. The PROTOCOL.ORA file must be created manually. See Appendix A in this manual and the Oracle operating system–specific documentation for your platforms for information about the contents of the PROTOCOL.ORA file.

Creating the TNSNAV.ORA File for Clients and Servers

If you do not have an Interchange in your network, you do not need to use the TNSNAV.ORA file, although Network Manager generates one automatically.

LOCAL_COMMUNITIES

This section is required in every TNSNAV.ORA file. It is simply a list of the communities of which the node is a member. The COMMUNITY_LIST keyword is required only if there is more than one community.

Format of TNSNAV.ORA

This file uses the keyword–value pair syntax described for the TNSNAMES.ORA file earlier in this chapter. Groups of keyword–value pairs that logically belong together are surrounded by parentheses.

Sample LOCAL_ COMMUNITIES Entries

For example, on a client that is a member of two communities, the LOCAL_COMMUNITIES entry in TNSNAV.ORA may appear as:

```
LOCAL_COMMUNITIES = (COMMUNITY_LIST =
                    (COMMUNITY = NMP.FIN.HQ.ACME)
                    (COMMUNITY = TCP.FIN.HQ.ACME))
```

For a client that is a member of just the Named Pipes community indicated, the following entries are equivalent:

```
LOCAL_COMMUNITIES = (COMMUNITY = NMP.FIN.HQ.ACME)
```

and

```
LOCAL_COMMUNITIES = (COMMUNITY_LIST =
                    (COMMUNITY = NMP.FIN.HQ.ACME))
```

Keywords used within PREFERRED_CMANAGERS

Within the PREFERRED_CMANAGERS section, one or more CMANAGERs are listed, specifying which Connection Managers should be used as the first hop for multi–community connections in order from most preferred to least preferred. The entry takes the form:

```
PREFERRED_CMANAGERS=[(CMANAGER_LIST=]
                (CMANAGER=
                   (CMANAGER_NAME=cmanager_name)
                   (ADDRESS=
                     (COMMUNITY=community_name)
                     (PROTOCOL=protocol_ID)
                     (protocol specific information)))
                [(CMANAGER=
                   (CMANAGER_NAME=cmanager_name)
                   (ADDRESS=
                     (COMMUNITY=community_name)
                     (PROTOCOL=protocol_ID)
                     (protocol specific information)))]
             [ )]
```

In this syntax:

CMANAGER_LIST	Indicates that one or more descriptions of a Connection Manager follow, each identified by the keyword CMANAGER. This keyword is preceded by a left parenthesis, and finished with a right parenthesis following the last Connection Manager described. Where only one CMANAGER follows, this keyword is optional.
CMANAGER	Indicates that a Connection Manager is described following the keyword. The CMANAGER_NAME and the TNS address of the associated Connection Manager follow. At least one CMANAGER must be defined for any client or server that uses an Interchange.
CMANAGER_ NAME	Indicates the name of a Connection Manager. (The Connection Manager and the Interchange are usually referred to by the same name.)
ADDRESS	Indicates the address of the Connection Manager named in the CMANAGER_NAME parameter. Note that the Connection Manager has its own address for receiving connections on the Interchange machine. This is not the same as the

address of a TNS listener if there is one on that machine.

Oracle Network Manager will supply the correct keywords and format for this configuration file; it will prompt you to supply the needed values for each CMANAGER_NAME and ADDRESS.

Sample TNSNAV.ORA File

The following sample TNSNAV.ORA file is for clients and servers in a DECnet community. There are two Interchanges available in this community.

```
LOCAL_COMMUNITIES =
  (COMMUNITY_LIST =
    (COMMUNITY = DECCOM.WORLD)
  )
PREFERRED_CMANAGERS =
  (CMANAGER_LIST =
    (CMANAGER =
      (CMANAGER_NAME = INT2.WORLD)
        (ADDRESS =
          (COMMUNITY = DECCOM.WORLD)
          (PROTOCOL = DECNET)
          (NODE = IRIS.WORLD)
          (OBJECT = CMAN)
        )
    )
    (CMANAGER =
      (CMANAGER_NAME = INT1.WORLD)
        (ADDRESS =
          (COMMUNITY = DECCOM.WORLD)
          (PROTOCOL = DECNET)
          (NODE = DAISY.WORLD)
          (OBJECT = CMAN)
        )
    )
  )
```

Further Examples

NetMan

For examples of all the configuration files required in a network using SQL*Net and the Oracle MultiProtocol Interchange, see Appendix B of *Oracle Network Manager Administrator's Guide*

APPENDIX

C

Common Error Messages

Ideally, your installation, configuration, and use of SQL*Net will go smoothly, and you will have no problems. However, inevitably errors will occur. All the error messages you may receive when using Oracle networking products are described in the *Oracle Network Products Troubleshooting Guide*. However, for your convenience, this appendix describes seven of the most common error messages in detail. They are:

- ORA–12154 "TNS:could not resolve service name"
- ORA–12198 "TNS:could not find path to destination"
- ORA–12203 "TNS:unable to connect to destination"
- ORA–12500 "TNS:listener failed to start a dedicated server process"
- ORA–12533 "TNS:illegal ADDRESS parameters"
- ORA–12545 "TNS:name lookup failure"
- ORA–12560 "TNS:protocol adapter error"

Diagnostics

For information on other error messages, and to learn more about error logging and the trace facility, see the *Oracle Network Products Troubleshooting Guide.*

Common Sources of Error: Configuration Files

The most common cause of problems in making a SQL*Net connection is in the structure or location of the configuration files.

Syntax Errors

The syntax of the files must be followed precisely. If you create the files using Network Manager, syntax errors should not occur. However, if you create or edit them by hand, check them carefully to see that all the appropriate parentheses are in place, that the lines are indented to show their logical structure, and that there are no typographical errors. See Appendix B, ”Syntax Rules for Configuration Files,” for details on the syntax of these files. Be sure that all service names are mapped to connect descriptors in the TNSNAMES. ORA file.

Location Errors

The TNSNAMES.ORA file must be distributed so that it is available to all nodes that make connections using SQL*Net version 2 (unless Oracle Names is being used on the network).

A unique LISTENER.ORA file must be distributed to each node that includes one or more databases.

In a network that includes one or more Interchanges, a unique TNSNAV.ORA file must be available to all the clients which have similar navigation requirements.

Every node with an Interchange must have an INTCHG.ORA, a TNSNET.ORA, and a TNSNAV.ORA file.

OS Doc

See the Oracle operating system specific manual for your platform for the exact locations of these files.

Common Sources of Error: Inactive Components

Before you can use a SQL*Net network, a transport layer protocol and an Oracle Protocol Adapter must be installed.

Be sure that you have started the listener for any server you intend to contact. For instructions on starting the listener using the Listener Control Utility, see Chapter 5, ‘‘Using SQL*Net”.


Intchg

If you are making a connection across protocols using one or more Interchanges, the Interchanges must be started. For instructions on starting an Interchange using the Interchange Control Utility, see Chapter 6 in the *Oracle MultiProtocol Interchange Administrator's Guide.*

Common Error Messages


Diagnostics

The following error messages are among those most often encountered by SQL*Net users. For a complete list of SQL*Net error messages, their causes and the action to take if you get them, see Chapter 3 in the *Oracle Network Products Troubleshooting Guide.*

ORA–12154 TNS:could not resolve service name

This error points to a problem related to Oracle Names or the TNSNAMES.ORA file. The service name specified cannot be located through either a Names Server or the TNSNAMES.ORA file. If you get this error, do the following:

- If you are using Oracle Names:
  - Use the Names Control Utility STATUS or PING commands to see if a Names Server is available. For example, enter the following:

    ```
    namesctl status names_server_name
    ```

  - Use the Names Control QUERY command to find out if the service name and address are stored on a Names Server.

    For example, enter the following:

    ```
    namesctl query service_name *
    ```


ONames

Refer to Appendix A in the *Oracle Names Administrator's Guide* for more detailed information.

- If you are using a TNSNAMES.ORA file:
  - Verify that a TNSNAMES.ORA file exists and is in the proper place and accessible. See the Oracle operating system-specific manual for your platform for details on the required name and location.


OS Doc

  - Check to see that the service name is mapped to a connect descriptor in the TNSNAMES.ORA file and add or correct it if necessary.

– Make sure there are no syntax errors anywhere in the file. Particularly look for unmatched parentheses or stray characters. Any error in a TNSNAMES.ORA file makes it unusable. See Appendix B, "Syntax Rules for Configuration Files," for details on the syntax of this file.

ORA–12198 TNS:could not find path to destination

This error indicates a failure of navigation across protocols through one or more Interchanges. If you get this error, do the following:

- Verify that the Interchanges are active. (Use the STATUS command of the Interchange Control Utility.) If necessary, turn on the Interchanges with the START command of the Interchange Control Utility (INTCTL). For information about how to use the Interchange Control Utility, see Chapter 6 of the *Oracle MultiProtocol Interchange Administrator's Guide*.

Intchg

- Verify that each Interchange has enough pump capacity to make another connection. (Use the STATUS command of INTCTL.) If not, wait and try to make the connection later.
- If the Interchanges seem to be functioning correctly, check the TNSNAMES.ORA file to verify that the correct community is included in the ADDRESS section of the connect descriptor of the service name you are using. Also check the client's TNSNAV.ORA file to be sure that the correct address is included for the preferred Connection Managers.

If the error persists, it may be useful to check for errors in the Interchange log files. A different error in the connection process can sometimes be reported as a 12198.

ORA–12203 TNS:unable to connect to destination

This message indicates that the client is not able to find the desired database. If you get this message, do the following:

- Be sure you have entered the service name you wish to reach correctly.
- Look at the TNSNAMES.ORA file to see that the ADDRESS parameters in the connect descriptor for the service name are correct.
- Use the Listener Control Utility to verify that the listener on the remote node is up and listening. If it is not, use the Listener Control Utility to start it. For instructions on starting the listener

using the Listener Control Utility, see Chapter 5, "Using SQL*Net."

This error often is reported because the destination node is not available or because of unrelated network problems.

ORA–12500 TNS:listener failed to start a dedicated server process

This message indicates that the listener was unable to start a process connecting the user to the database server. The most likely cause is that the SID_LIST section of the LISTENER.ORA file or the system identifier (SID) in the CONNECT DATA section of the TNSNAMES.ORA file is incorrect. Check each of these files.

Another possible cause of this message is that the user does not have adequate privileges to access the database. Ask your database administrator to provide you access privileges if you do not have them. Also ask your system administrator to check the ownership and permissions of the database server process image.

ORA–12533 TNS:illegal ADDRESS parameters

OS Doc

This message occurs if the protocol specific parameters in the ADDRESS section of the designated connect descriptor in TNSNAMES.ORA are incorrect. The protocol specific keywords, and their acceptable values, can be found in the Oracle operating specific documentation for your platform.

If you use Network Manager to create the TNSNAMES.ORA file, the correct protocol specific keywords are included automatically.

ORA–12545 TNS:name lookup failure

This message occurs when the listener on the remote node cannot be contacted. The ADDRESS in the TNSNAMES.ORA file or the LISTENER.ORA file may be incorrect. This message may also appear if the listener on the remote node has not been started. Check its status with the STATUS command of the Listener Control Utility, and start it with the START command if necessary.

ORA–12560 TNS:protocol adapter error

This message indicates that there is a problem at the protocol adapter level. That is, SQL*Net and the Interchange are functioning correctly, but there is something wrong with the protocol adapter that underlies

OS Doc

them. The Oracle operating system–specific documentation for your platform contains more information about the protocol adapters.

In order to get more information about the specific problem, turn on tracing and re-execute the operation. (Because tracing and trace files use a lot of disk space, be sure to turn off tracing as soon as the trace is complete.) The trace file includes information about the specific protocol adapter problem. If you have trouble interpreting the trace file, contact Worldwide Customer Support for assistance, following the procedures described in Chapter 1 of the *Oracle Network Products Troubleshooting Guide*.

Diagnostics

APPENDIX

D SQL*Net OPEN

This appendix describes the new SQL*Net feature, SQL*Net OPEN. SQL*Net OPEN is the application program interface (API) to SQL*Net. Using SQL*Net OPEN, programmers can develop both database and non–database applications that make use of the SQL*Net network already deployed in their environment.

By exposing an interface to SQL*Net, SQL*Net OPEN provides applications a single common interface to all industry standard network protocols. With SQL*Net, database applications can connect to other database applications across networks with different protocols. SQL*Net OPEN extends this connectivity to non–database applications.

SQL*Net OPEN provides a simple and portable call interface that enables developers to build truly portable network applications. They can rapidly deploy an application developed on one machine to another without having to modify their calls to the network interface.

> **Note:** SQL*Net OPEN in SQL*Net release 2.3.2 is in beta status. A production version with expanded functionality will be available with release 2.3.3.

The relationship of SQL*Net OPEN to other products is shown in Figure D – 1.

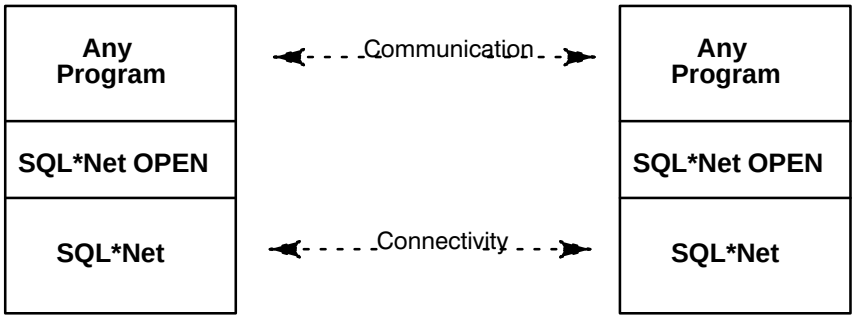

Figure D – 1 SQL*Net OPEN

Practical Uses for SQL*Net OPEN

You can use SQL*Net OPEN to solve a number of problems, such as:

- Client / agent / server connectivity

  SQL*Net OPEN enables you to use any application to communicate with any agent. For example, the agent might be an application server that allows simultaneous connectivity to Oracle and non–Oracle data sources, such as remote information servers.

- Distributed applications

  Using SQL*Net OPEN, you can build distributed applications that do not require a database and that can run over an existing Oracle network without any additional middleware.

- Enhanced clients

  You can use SQL*Net OPEN to integrate non–SQL information into SQL applications. For example, a process control application could communicate with a non–SQL application such as a sensor.

Compatibility with Oracle Network Products

SQL*Net OPEN is compatible with all releases of SQL*Net 2.0 and later. It works with related Oracle network products.

Oracle MultiProtocol Interchange

Using SQL*Net OPEN, you can take advantage of the cross–protocol connectivity provided by Oracle MultiProtocol Interchange. Through the

MultiProtocol Interchange, applications can communicate across networks based on different protocols.

No additional programming is required to take advantage of the MultiProtocol Interchange. Its transfer of data across protocols is transparent to both users and programmers.

Secure Network Services

You can protect all your network data with standard encryption and data integrity checking through Secure Network Services. Again, no additional programming is required to take advantage of these services.

SQL*Net OPEN Function Calls

The SQL*Net OPEN API consists of four function calls to perform the simple open, send, receive, and close functions common with network applications. Future versions of the API will add a ”control” call to provide access to more advanced features of the SQL*Net transport.

Enhanced functionality will be available in future releases.

SQL*Net OPEN API Usage

SQL*Net OPEN provides a byte stream-oriented API which can be used to develop basic applications which send and receive data. This can be contrasted with a remote procedure call interface, for example, which provides the abstraction of a procedure call, and handles the marshalling of input and output arguments accordingly. Applications developed with SQL*Net OPEN must ensure for themselves that values that are sent across the network will be interpreted correctly at the receiving end.

The API as currently provided supports the C language. Examples given in this documentation are in C. Names used in API begin with ”TNS,” which stands for Transparent Network Substrate, the network transport technology on which SQL*Net version2 is based. For more information about TNS, refer to Chapter 2 in this manual.

Finding the SQL*Net OPEN API

The API is provided as part of the standard SQL*Net installation. To use the API, you need:

- tnsapi.h, the header file which describes the API interfaces and errors. This file is provided in the appropriate location under the Oracle home directory for your platform. On UNIX, it is provided in $ORACLE_HOME/network/include.
- The SQL*Net OPEN library. The name of the library varies by platform, but it is located where the other Oracle networking libraries are located, and contains the name "TNSAPI". On UNIX, it is in your $ORACLE_HOME/lib directory and is named libtnsapi.a. On Windows, the Oracle directories contain the files TNSAPI.DLL and TNSAPI.LIB in the appropriate locations.
- Sample makefiles. These are provided for your platform under your network directory, as above. They can be used to determine the appropriate link line to use to build your application.

Building Your Own Application

Modules which make reference to SQL*Net OPEN functions should include the file tnsapi.h, as follows:

```
#include <tnsapi.h>
```

Your makefile (or other location for your build command) should ensure that the include path is set properly so that tnsapi.h will be found. Refer to the sample makefiles provided in your installation.

The SQL*Net OPEN API functions are described in the next sections:

TNSOPEN

Synopsis

```
int tnsopen(handlep, name)
void **handlep;
const char *name;
```

Description

Initializes the SQL*Net OPEN API per–connection handle. This function must be the first SQL*Net OPEN call that a user makes. Note that tnsopen() does not establish a connection; connections are established by the send and receive calls as needed. See the sections on tnssend() and tnsrecv() below.

If you're writing a client program (which will initiate the connection), "name" contains a service name in the same format as those in the SQL*Net configuration file TNSNAMES.ORA.

If you're writing a server program, "name" should be NULL. The connection will automatically be picked up by your server program when you make the first tnsrecv() call to receive data (see the section on tnsrecv, below.)

Parameters

`handlep (IN/OUT)` – Address to receive SQL*Net connection handle
`name (IN)` – service name

Prerequisites

The handlep parameter must not be NULL

Returns

Upon successful completion a zero value is returned. Otherwise, a positive SQL*Net API error is returned, as listed in the SQL*Net OPEN API Errors section.

TNSCLOSE

Synopsis

```
int tnsclose(handlep)
void **handlep;
```

Description

Shuts down the connection. This function must be called by the user to close the connection and release the handle properly.

Parameters

`handlep(IN/OUT)` – Address of a pointer to a SQL*Net connection handle

Prerequisites

The handlep parameter must not be NULL.

Returns

Upon successful completion a zero value is returned, and *handlep is set to NULL. Otherwise, a positive SQL*Net API error number is returned.

TNSSEND

Synopsis

```
int tnssend(handle, data, length)
void *handle;
const void *data;
size_t *length;
```

Description

Send data to the SQL*Net connection handle.

In the first call to tnssend() on the client side, the connection is established before any data is sent to the handle. The client application must first call tnssend() after tnsopen() to establish a connection to the server. It is an error if the client application calls tnsrecv() first, or if the server program calls tnssend() first.

Note that this also means that the tnssend() call may return errors related to connection establishment – so the first indication you get that, for instance, you've given the incorrect TNS address, is that an error occurs on the first tnssend() call, rhater than on the tnsopen() call as you may first expect.

Parameters

`handle(IN/OUT)` – pointer to SQL*Net connection handle returned by tnsopen()
`data(IN)` – pointer to data to be sent
`length(IN/OUT)` – pointer to the length of data to be sent in bytes and the actual number of bytes written on return.

Prerequisites

The parameters must not be NULL.

Returns

Upon successful completion a zero value is returned, and the actual number of bytes written is returned as the value pointed to by the length parameter. Otherwise, a positive SQL*Net API error number is returned.

TNSRECV

Synopsis

```
int tnsrecv(handle, data, length)
void *handle;
void *data;
size_t *length;
```

Description

Receive data from the SQL*Net connection handle.

In the first call to tnsrecv() on the server side, the connection is established before data is received from the handle. The server must first call tnsrecv() after tnsopen() to accept the connection from the client.

Incoming connections are accepted by the SQL*Net Listener (tnslsnr), which automatically spawns off a copy of your server program when needed, and hands it the new connection. You must configure the network listener with the location of your server program for this to occur — see the section on configuration below.

Parameters

`handle(IN/OUT)` – pointer to SQL*Net connection handle returned by tnsopen()
`data(IN/OUT)` – pointer to buffer to receive data
`length(IN/OUT)` – pointer to the length of buffer to receive data and actual number of bytes received on return

Prerequisites

All parameters must not be NULL.

Returns

Upon successful completion a zero value is returned, and the actual number of bytes received is returned as the value pointed to by the length parameter. Otherwise, a positive SQL*Net API error number is returned.

SQL*Net OPEN API Errors

This section lists the error numbers which can be returned if one of the above function calls fails. Note that in some cases, connection–related errors may come back from a send or receive call, if the connection has not yet been established at that time.

```
20002 – SDFAIL_TNSAPIE – The underlying ”send” command failed in
tnssend().
20003 – RECVFAIL_TNSAPIE – The underlying ”receive” command failed
in tnsrecv().
20004 – INVSVROP_TNSAPIE – Operation is invalid as the server.
20005 – INVCLIOP_TNSAPIE – Operation is invalid as the client.
20006 – HDLUNINI_TNSAPIE – The connection should be initialized by
calling tnsopen().
20007 – INHFAIL_TNSAPIE – Server failed in inheriting the
connection from the listener.
20008 – ACPTFAIL_TNSAPIE – Server failed in accepting the
connection request from the client.
20009 – NULHDL_TNSAPIE – A null handle was passed into the call,
which is not allowed.
20010 – INVOP_TNSAPIE – An invalid operation called was passed
into the call.
20011 – MALFAIL_TNSAPIE – A malloc failed in TNS API call.
20012 – NLINIFAIL_TNSAPIE – Failed in NL initialization.
20013 – NMTOOLONG_TNSAPIE – Service name is too long.
20014 – CONFAIL_TNSAPIE – Client connect request failed.
20015 – LSNFAIL_TNSAPIE – Server failed to listen for connect
request.
20016 – ANSFAIL_TNSAPIE – Server failed to answer connect request.
20017 – NMRESFAIL_TNSAPIE – Failed to resolve service name.
20020 – TNSAPIE_ERROR – TNS error occurred.
```

Configuring the System to Use Your Application

In this release of SQL*Net OPEN, Oracle Network Manager cannot be used to configure applications you have developed using this API. You must configure SQL*Net yourself to recognize your application. This involves three steps:

1. Add the location of your server program to LISTENER.ORA, so that the network listener knows to start up your server if a connection request comes in for your service — rather than starting up an Oracle database server.

To do this, first you must come up with a SID name for your service, similar to the one an Oracle database has. (Don't pick the same SID as your database.)

For example, say you are configuring a "chat" program. You could call the SID "chatsid". Place the program into an existing ORACLE_HOME, which we'll assume here is in /usr/oracle.

You would place the following entry in the LISTENER.ORA file for your listener:

```
SID_LIST_LISTENER =
  (SID_LIST =
    (SID_DESC =
      (SID_NAME = chatsid)                /*your SID name*/
      (ORACLE_HOME = /usr/oracle)         /*your ORACLE_HOME name*/
      (PROGRAM = chatsvr)                 /*the name of your server
                                                  program*/
    )
```

Note: There may be additional SIDs in the list as well, configured with Network Manager.

You need to restart the listener so it will recognize the new service.

2. Add the address of your application server to TNSNAMES.ORA, just like Network Manager would for a database server.

   Let's assume that your listener is on the address:

```
(DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=unixhost)(PORT=1521)))
```

   Also assume that you want people to refer to the service you created above as "chat".

   You would add to the TNSNAMES.ORA file:

```
chat=(DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=unixhost)
  (PORT=1521))(CONNECT_DATA=(SID=chatsid)))
```

   Note that the address contains the SID you configured in the LISTENER.ORA file above. Also note that the second line started with at least one space character, which indicates that it is a continuation line. In general, if you are editing a file that was originally created with Oracle Network Manager, you should follow its formatting conventions.

   If you have a default domain, you need to name your service accordingly. For instance, chat.acme.com if your default domain is acme.com. Again, use the existing TNSNAMES.ORA file as a template — if all the other service names end in a domain, you need to name your service name similarly.

3. Place the executable for your service in the appropriate location so the network listener can see it.

   Your server program needs to go into the directory where your Oracle Server executable is. This is specific to your particular platform. On UNIX systems, it should go into the ORACLE_HOME directory that you indicated in the LISTENER.ORA file. So, in this example, you would place the program "chatsvr" in the location /usr/oracle/bin/chatsvr.

   You also must ensure that your program has the correct permissions to be executed, if this is needed on your operating system.

Sample Programs

In this release, two sample applications are provided with your SQL*Net OPEN distribution. They are:

- finger – The finger utility connects to a server which retrieves information about who is logged in. This is very simple pair of programs which demonstrate the basic steps involved in building a distributed application. The client program runs on both Solaris and Windows NT; the server is UNIX specific.
- tftp – A client and server program are included which implement simple file transfer using the tftp protocol. This is a more extensive example. This demo was implemented for UNIX.

Glossary

address A unique network location used to identify a network object, such as a database server, client, MultiProtocol Interchange, or Names Server. TNS addresses have a specific format. Addresses must be unique. See *TNS address*.

administrative region An organizational entity for administering SQL*Net network components.

alias An alternative name for an existing network object.

ASCII character set Stands for American Standard Code for Information Interchange character set, a convention for representing alphanumeric information using digital data. The collation sequence used by most computers with the exception of IBM and IBM–compatible computers. Contrast with *EBCDIC character set*.

buffer (1) A temporary storage area for data. (2) In SQL*Net, an area of memory used by the network driver to pass data between two points on the network. (3) In SQL*Plus, an area where the user's SQL statements or PL/SQL blocks are temporarily stored. SQL*Plus buffers are supplied for users' convenience in working with multiple statements. The SQL buffer is the default buffer. You can use multiple buffers.

cache The in–memory database within a Names Server where all data is stored.

central administration A SQL*Net network in which all data is administered in one administrative region.

client A user, software application, or computer that requests the services, data, or processing of another application or computer. In a two–task environment, the client is the user process. In a network environment, the client is the local user process and the server may be local or remote.

client profile The properties of a client, often shared by many clients. Includes the protocols used, preferred Interchanges, and preferred Names Servers. Also called *client type*.

client type See *client profile*.

client–server architecture Software architecture based on a separation of processing between two CPUs, one acting as the client in the transaction, requesting and receiving services, and the other as the server that provides services in a transaction.

community A group of network clients and servers using TNS–based software that can communicate using the same industry–standard protocol.

community cost A number assigned to a community to reflect its relative communications speed and throughput. The network administrator assigns costs in the file TNSNET.ORA. The cost is used by the Navigator when selecting the best path to a destination.

community name A unique name used to identify a TNS community in a TNS network.

configuration files Files that are used to identify and characterize the components of a network. Configuration is largely a process of naming network components and identifying relationships among those components.

CONNECT DATA A portion of the connect descriptor, introduced by the keyword CONNECT DATA, that specifies the application to which the connection is to be made. The CONNECT DATA section of the connect descriptor includes the *Oracle System Identifier* (SID).

connect descriptor A specially formatted description of the destination for a network connection. Connect descriptors are constructed using a set of keywords and values. They are mapped to *service names* to provide more convenient reference.

connection An interaction between two clients on a network. *TNS connections* are originated by an *initiator* (client), who requests a connection with a *destination* (server).

Connection Manager The component of the MultiProtocol Interchange that detects, establishes, and maintains data connections over a TNS network. The Connection Manager contains a *listener* and multiple *data pumps*.

connection request A notification sent by an initiator and received by a listener that indicates that the initiator wants to start a TNS connection.

cost See *community cost*.

database administrator (DBA) (1) A person responsible for operating and maintaining an Oracle Server or a database application. (2) An Oracle username that has been given DBA privileges and can perform database administration functions. Usually the two meanings coincide. Many sites have multiple DBAs.

database link A network object stored in the local database or in the network definition that identifies a remote database, a communication path to that database, and optionally, a username and password. Once defined, the database link is used to access the remote database. Also called DBlink.

A public or private database link from one database to another is created on the local database by a DBA or user.

A global database link is created automatically from each database to every other database in a network with Oracle Names. Global database links are stored in the network definition.

database service name See s*ervice name*.

default domain The domain within which most client requests take place. It could be the domain where the client resides, or it could be a domain from which the client requests network services often. *Default domain* is also the client configuration parameter that determines what domain should be appended to unqualified network name requests. A name request is unqualified if it does not have a "." character within it.

delegated administration A SQL*Net network where network management is delegated to one or more administrative regions below the root administrative region. Each region is administered through an installation of Oracle Network Manager. Also referred to as distributed or decentralized administration.

delegated administrative region A region hierarchically below the root administrative region. Any region other than the root administrative region.

destination The client that is the endpoint of a TNS connection. The initiator of the connection requires some data or service of the destination.

distributed architecture A blueprint or specification that allows applications to access data on more than one CPU (and often many types of computers ranging from PCs to mainframes) within a computer system or network.

distributed database One logical database whose data is spread across two or more computers. Users interact with the single, logical database, not the individual pieces. See also *distributed processing.*

distributed processing Division of front–end and back–end processing to different computers. SQL*Net supports distributed processing by transparently connecting applications to remote databases.

distributed query A query that selects data from multiple databases, using, for example, joins, nested queries, or views. See also *distributed database.*

domain A grouping of network objects, such as databases, that simplifies the naming of network services. Within a domain, all the names must be unique.

domestic domains The set of domains that are managed within a given administrative region. Domains are only domestic relative to a region; they are not domestic in any absolute sense. Also referred to as *local* domains.

Dynamic Discovery Option An optional feature of Oracle Names version 2.0 that enables services to register themselves with well–known Names Servers on startup, enables clients to find well–known Names Servers without configuration, and enables well–known Names Servers to automatically replicate their data to each other.

error message A message from a computer program informing you of a potential problem or condition preventing program or command execution.

flat naming model An Oracle Names infrastructure in which there is only one domain, WORLD. All names must be unique within the WORLD domain.

foreign domains The set of domains not managed within a given administrative region. Domains are only foreign relative to a region; they are not foreign in any absolute sense.

global database link A database link created automatically by Network Manager for use with Oracle Names that links each database in a network to all other databases.

global database name A unique name that identifies a database in a network. It consists of a database name and its network domain.

hierarchical naming model A network naming scheme in which names are divided into multiple hierarchically related domains.

hop A portion of a connection path. It may be between two communities, between two Interchanges, or between an Interchange and the destination. Or, the path may simply have one hop, from the initiator directly to the destination.

initiator The client that starts a TNS connection by sending a connection request. The initiator of the connection requires some data or service of the destination.

INTCHG.ORA file A file maintained for each Interchange that defines the startup parameters of an Interchange.

Interchange Control Utility (INTCTL) A utility included with the MultiProtocol Interchange to control various functions, such as to start, stop, and gather statistics from the Interchange.

keyword–value pair The combination of a standard TNS keyword and a value, used as the standard unit of information in connect descriptors and many configuration files. Keyword–value pairs may be nested; that is, a keyword may have another keyword–value pair as its value.

listener process The server process that listens for and accepts incoming connection requests from client applications. Oracle listener processes start up Oracle database processes to handle subsequent communications with the client.

listener See *network listener*

LISTENER.ORA A configuration file that describes one or more TNS listeners on a server.

Listener Control Utility (LSNRCTL) A utility included with SQL*Net version 2 to control various functions, such as to start, stop, and get the status of the network listener.

logging A feature in which errors, service activity, and statistics are written to a log file. See also *tracing*.

location transparency A distributed database characteristic that allows applications to access data tables without knowing where they reside. All data tables appear to be in a single database, and the system determines the actual data location based on the table name. The user can reference data on multiple nodes in a single statement, and the system automatically and transparently routes (parts of) SQL statements to remote nodes for execution if needed. The data can move among nodes with no impact on the user or application.

multicommunity connection A connection between an initiator and a destination that belong to different communities. The connection passes between at least two communities, by way of one or more Interchanges.

MultiProtocol Interchange An Oracle networking product that allows applications in TNS networks to communicate across different protocols. For example, a TCP/IP client could connect through an Interchange to an Oracle7 server running only DECnet.

naming model The set and structure of domains within which names can be allocated.

In a flat naming model, there is a single domain — the WORLD domain. Examples of database service names in a flat naming model are SALES.WORLD or HR.WORLD.

In a hierarchical naming model, the highest level is the root domain, and all other domains are hierarchically related. An example of a service name in a hierarchical naming model is HR.US.ORACLE.COM.

native naming adapters Support for industry–standard name services such as Network Information Services (NIS), Distributed Computing Environment Cell Directory Service (DCE CDS), Banyan StreetTalk, and Novell's NetWare Directory Services (NDS) enables users to connect to Oracle services in various native naming environments.

Navigator The component of an Interchange that chooses (navigates) the best connection path over a TNS network. In order to navigate, the Navigator refers to the file TNSNET.ORA.

negotiation A communication between two processes in which they agree upon how they will communicate. In a TNS connection, a negotiation occurs between the initiator and the destination, and at every intermediate hop.

NetBIOS Stands for Network Basic Input/Output System, a process–to–process communications protocol designed to provide virtual links between machines on a network.

NetWare A network operating system produced by Novell.

network A group of two or more computers linked together through hardware and software to allow the sharing of data and/or peripherals.

network administrator The person who performs network management tasks such as installing, configuring, and testing network components. The administrator typically maintains the configuration files, TNS connect descriptors and service names, aliases, and public and global database links.

network character set As defined by Oracle, the set of characters acceptable for use as values in keyword–value pairs (that is, in connect descriptors and configuration files). The set includes alphanumeric upper– and lowercase, and some special characters.

network data Includes all communities and MultiProtocol Interchanges in the entire network, and all Names Servers in the root or central administrative region. Also called *backbone* data.

network definition The network configuration created by Network Manager. When you enter and save information in Network Manager, it creates a network definition in an operating system file or a database. It generates configuration files from the network definition, and uses it as a data source for Oracle Names.

network listener A listener on a server that listens for connection requests for one or more databases on one or more protocols.

Network Manager The graphical tool for configuring and maintaining a SQL*Net network, including SQL*Net, the MultiProtocol Interchange, and Oracle Names.

network object Any service that can be directly addressed on a network; for example, a listener, MultiProtocol Interchange, or Names Server.

network protocol Any industry standard transport protocol, such as TCP/IP or APPC.

network service In an Oracle application network, a service performs tasks for its service consumers; for example, a MultiProtocol Interchange provides protocol conversion services for clients, and a Names Server provides name resolution services for clients.

node A computer or terminal that is part of a TNS network.

ORACLE_HOME An alternate name for the top directory in the Oracle directory hierarchy on some directory–based operating systems.

Oracle Names A name resolution product for Oracle7 Servers and SQL*Net networks.

Oracle Names Server (Names Server) The name resolution software for answering name requests.

Oracle Protocol Adapters A set of products in which each one maps the functions of a given network protocol into TNS, so that TNS can act as an interface among all protocols. For example, there is a TCP/IP protocol adapter and a DECnet adapter. SQL*Net version 2 requires protocol adapters; the equivalent in SQL*Net version 1 was the protocol driver component of SQL*Net.

Oracle System Identifier (SID) A unique name for an Oracle database instance. To switch between Oracle databases, users must specify the desired SID. The SID is included in the CONNECT DATA parts of the connect descriptors in a TNSNAMES.ORA file, and in the definition of the network listener in the LISTENER.ORA file.

parameter Information passed to a program, command, or function, such as a file specification, a keyword, or a constant value.

password A string (word or phrase) used for data security and known only to its owner. Passwords are entered in conjunction with an operating system login ID, Oracle username, or account name, in order to connect to an operating system or software application (such as the Oracle database). Whereas the username or ID is public, the secret password ensures that only the owner of the username can use that name, or access that data.

path A series of nodes traversed by a TNS connection, to send data from one end of a connection to the other. A path is made up of hops, which are addresses of the next points on the path.

path correction The process by which a Navigator determines that a particular connection is taking a less than optimal path, and so redirects the connection so that it follows the best path.

preferred Connection Manager A Connection Manager, or Interchange, that has been identified by a particular TNS client as the best starting point for navigating to that client's destinations. Preferred Connection Managers are identified in each client's TNSNAV.ORA file.

preferred Names Server The Names Server preferred by a client for names resolution.

prespawned dedicated server process Prespawned dedicated server processes are prestarted by the SQL*Net listener before any incoming connection request. They improve the time it takes to establish a connection on servers where the MultiThreaded Server is not used or not supported on a given machine. They also use allocated memory and system resources better by recycling server processes for use by other connections with shutting down and recreating a server. The ability to prespawn dedicated servers requires Oracle7 Server release 7.1 and SQL*Net release 2.1 or later.

private database link A DBlink created by one user for his or her exclusive use. See also *database link*.

protocol adapter See *Oracle Protocol Adapters*.

public database link A database link created by a database administrator on a local database which is accessible to all users on that database. See also *database link*.

pump The Interchange component responsible for transferring data between two communities as a part of a TNS connection. The Interchange's Connection Manager controls several data pumps, starting them as required.

root domain The highest level domain in a hierarchical naming model.

root administrative region The highest level administrative region in a distributed installation. The root administrative region contains the root domain.

service name A name for a TNS connect descriptor that is easy to use and remember. End users need only know the appropriate service name to make a TNS connection. Each connect descriptor is assigned a service name in the network definition. The service name for a database must be the same as the global database name.

SID See *Oracle System Identifier*.

single community connection A connection between an initiator and a destination that belong to the same community. The connection is confined to one community and thus a single network protocol.

SPX Stands for Sequenced Packet Exchange, a network protocol known for high performance and acceptance among many major network management systems, in particular, Novell Advanced NetWare.

SQL*Net An Oracle product that works with the Oracle Server and enables two or more computers that run the Oracle RDBMS or Oracle tools such as SQL*Forms to exchange data through a third–party network. SQL*Net supports distributed processing and distributed database capability. SQL*Net runs over many communications protocols.

SQL*Net OPEN An application program interface (API) to SQL*Net, which enables programmers to develop non–database applications that can use the SQL*Net network.

TNS See *Transparent Network Substrate*.

TNS address An address of a client or network service in a TNS network, which identifies the community to which it belongs, the protocol used by that community, and some protocol specific values. One TNS client or network service may have multiple TNS addresses if it resides in multiple communities.

TNS client A client node that is a member of one or more TNS communities and is running software based on TNS. A TNS client can be either a client, a server, or both.

TNS community See *community*.

TNS connection A connection initiated by a TNS client using a connect descriptor, and maintained by TNS and, optionally, a MultiProtocol Interchange.

TNSNAMES.ORA file A file that contains connect descriptors mapped to service names. The file may be maintained centrally or locally, for use by all or individual clients.

TNSNAV.ORA file There are two versions of this file. The TNSNAV.ORA file for a client specifies that client's preferred Connection Managers. Clients can share the file if they have identical network requirements (that is, if they are the same *client profile*). The TNSNAV.ORA file for an Interchange lists the communities of which that Interchange is a member. Each Interchange in a network must have its own TNSNAV.ORA file.

TNSNET.ORA file A file that includes a list of every Interchange on the network and the communities they connect. The file is used by Navigators to navigate connection paths. It is maintained centrally and copied as necessary. Every Interchange must be able to access this file.

TNS network A network composed of one or more TNS communities. Many TNS networks have one or more MultiProtocol Interchanges to connect communities based on different network protocols.

tracing A facility that writes detailed information about an operation to an output file. The trace facility produces a detailed sequence of statements that describe the events of an operation as they are executed. Administrators use the trace facility for diagnosing an abnormal condition; it is not normally turned on.

Transparent Network Substrate (TNS) A foundation technology, built into SQL*Net version 2, the MultiProtocol Interchange, and Oracle Names, that works with any standard network transport protocol.

username The name by which a user is known to the Oracle Server and to other users. Every username is associated with a password, and both must be entered to connect to an Oracle database.

Index

Symbols

Numbers

A

B

C

D

E

F

G

M

N

O

P

Q

R

S

T

U

V

W

Reader's Comment Form

Name of Document: Understanding SQL*Net®
Part No. A42484–1

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 topic, chapter, and page number below:

_____

_____

_____

_____

_____

_____

_____

_____

Please send your comments to:

Oracle Network Products Documentation Manager
Oracle Corporation
500 Oracle Parkway
Redwood City, CA 94065 U.S.A.
Fax: (415) 506–7200

If you would like a reply, please give your name, address, and telephone number below:

_____

_____

_____

Thank you for helping us improve our documentation.