Oracle Rdb7™

Guide to Using the Oracle SQL/Services™ Client API

Release 7.0

Part No. A41981-1

ORACLE®

Guide to Using the Oracle SQL/Services Client API

Release 7.0

Part No. A41981-1

The information contained in this document is subject to change without notice. If you find any problems in the documentation, please report them to us in writing. Oracle Corporation does not warrant that this document is error free.

Restricted Rights Legend Programs delivered subject to the DOD FAR Supplement are ′commercial computer software′ and use, duplication and disclosure of the programs shall be subject to the licensing restrictions set forth in the applicable Oracle license agreement. Otherwise, programs delivered subject to the Federal Acquisition Regulations are ′restricted computer software′ and use, duplication and disclosure of the programs shall be subject to the restrictions in FAR 52.227-14, Rights in Data—General, including Alternate III (June 1987). Oracle Corporation, 500 Oracle Parkway, Redwood City, CA 94065.

The programs are not intended for use in any nuclear, aviation, mass transit, medical, or other inherently dangerous applications. It shall be the licensee's responsibility to take all appropriate fail-safe, back up, redundancy and other measures to ensure the safe use of such applications if the programs are used for such purposes, and Oracle disclaims liability for any damages caused by such use of the programs.

Contents

3 Sample Application Guidelines

4 Performance Considerations

5 Logging for Performance and Debugging

6 API Routines

7 Data Structures

8 Data Types

A Deprecated and Obsolete Features

Index

Figures

Tables

Send Us Your Comments

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.

You can send comments to us in the following ways:

- Electronic mail — `nedc_doc@us.oracle.com`
- FAX — `603-897-3334 Attn: Oracle Rdb Documentation`
- Postal service

```
Oracle Corporation
Oracle Rdb Documentation
One Oracle Drive
Nashua, NH  03062
USA
```

If you like, you can use the following questionnaire to give us feedback. (Edit the online release notes file, extract a copy of this questionnaire, and send it to us.)

Name ______________________ Title ______________________

Company ______________________ Department ______________________

Mailing Address ______________________ Telephone Number ______________________

____

Book Title ______________________ Version Number ______________________

- 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 chapter, section, and page number (if available).

Preface

Oracle Rdb™ is a general-purpose database management system based on the relational data model.

Oracle SQL/Services™, a client/server component of Oracle Rdb, enables a client application program invoked on a client computer running on a supported operating system or transport, to access Oracle Rdb databases and other databases supported by SQL on a Digital UNIX or OpenVMS server system. See the overview chapter for a complete list of supported clients.

This manual describes how to develop Oracle SQL/Services client application programs.

Intended Audience

This manual is written primarily for experienced applications programmers; however, some sections are intended for the system manager responsible for maintaining and fine-tuning Oracle SQL/Services. Both programmers and system managers should read Chapter 1 for a recommended approach to the material in this guide and a discussion of the pertinent sections. In addition, system managers should refer to the *Oracle SQL/Services Installation Guide*, which provides information important to the installation of an Oracle SQL/Services system, and to the *Oracle SQL/Services Server Configuration Guide*, which provides information important to the configuration and maintenance of an Oracle SQL/Services system.

Operating System Information

You can find information about the versions of the operating system and optional software that are compatible with this release of Oracle Rdb and Oracle SQL/Services in the *Oracle Rdb7 Installation and Configuration Guide* and the *Oracle SQL/Services Installation Guide*, and also in the *Oracle Rdb7 Release Notes* and the *Oracle SQL/Services Release Notes*.

Contact your Oracle® Corporation representative if you have other questions about compatibility.

Structure

This manual contains the following chapters and appendix.

Chapter 1	Introduces Oracle SQL/Services. Provides a reading path for programmers and system managers.
Chapter 2	Provides a condensed discussion of dynamic SQL, API routines, Oracle SQL/Services data structures, recommendations for API development, and API application linking.
Chapter 3	Provides guidelines for application development using the Oracle SQL/Services sample application.
Chapter 4	Explains how to enhance application performance.
Chapter 5	Describes execution logging and how to use it for debugging and monitoring application performance.
Chapter 6	Presents detailed reference descriptions of the Oracle SQL/Services API routines.
Chapter 7	Presents detailed reference descriptions of the Oracle SQL/Services data structures.
Chapter 8	Describes the data types used in Oracle SQL/Services.
Appendix A	Lists and describes the deprecated features for Oracle SQL/Services V4.1 and higher.

Related Manuals

For more information, see the other manuals in this documentation set, especially the following:

- *Oracle Rdb7 Guide to SQL Programming*
- *Oracle Rdb7 SQL Reference Manual*
- *Oracle Rdb7 Release Notes*
- *Oracle SQL/Services Release Notes*
- *Oracle Rdb7 Installation and Configuration Guide*
- *Oracle SQL/Services Installation Guide*
- *Oracle SQL/Services Server Configuration Guide*

The *Oracle SQL/Services Release Notes* and the *Oracle Rdb7 Release Notes* are provided only as part of the software kit. PostScript and .TXT source files for these release notes are available in SYS$HELP on OpenVMS systems and in /usr/lib/dbs/sqlsrv/v70/doc on Digital UNIX systems.

Conventions

In this manual, Oracle Rdb refers to Oracle Rdb for OpenVMS and Oracle Rdb for Digital UNIX software. Release 7.0 of Oracle Rdb software is often referred to as V7.0.

The SQL interface to Oracle Rdb is referred to as SQL. This interface is the Oracle Rdb implementation of the SQL standard ANSI X3.135-1992, ISO 9075:1992, commonly referred to as the ANSI/ISO SQL standard or SQL92.

Oracle ODBC Driver for Rdb software is referred to as the ODBC driver.

OpenVMS means both the OpenVMS Alpha and the OpenVMS VAX operating systems.

The following conventions are also used in this manual:

. . .	Vertical ellipsis points in an example mean that information not directly related to the example has been omitted.
[ ]	Brackets enclose optional clauses from which you can choose one or none.
$	The dollar sign represents the DIGITAL Command Language prompt in OpenVMS and the Bourne shell prompt in Digital UNIX.
%	The percent sign represents the Digital UNIX default user prompt.
boldface text	Boldface type in text indicates a term defined in the text.
e, f, t	Index entries in the printed manual may have a lowercase e, f, or t following the page number; the e, f, or t is a reference to the example, figure, or table, respectively, on that page.

Terminology

Some Oracle SQL/Services terminology in this manual has changed effective with release 7.0 of Oracle SQL/Services. The revised terminology reflects changes in the current server architecture compared to the original Oracle SQL/Services architecture, and makes terminology consistent on Digital UNIX and OpenVMS platforms.

- The term *dispatcher* replaces all instances of the phrase *communications server*.
- The term *executor* or *executors* replaces all instances of the phrase *execute server process* or *execute server processes*.
- The term *service* replaces all instances of the term *class*.

- The term *universal* replaces all instances of the term *generic* when referring to a service type. For example, the phrase *session reusable generic service* becomes *session reusable universal service* and *generic service* becomes *universal service*.

These terminology changes are made throughout this manual.

Technical Changes and New Features

This section lists some of the new and changed features described in this manual since it was last revised with Version 6.1. The *Oracle SQL/Services Release Notes* and *Oracle Rdb7 Release Notes* provide information on all the new features and technical changes included in release 7.0. The major new features described in this manual include the following:

- New server on OpenVMS systems with a server system management command line interface to manage the server components from OpenVMS and Digital UNIX server systems and an Oracle SQL/Services Manager graphical user interface (GUI) to manage server components from Windows clients. See the *Oracle SQL/Services Server Configuration Guide* and the *Oracle SQL/Services Installation Guide* for more information.
- Support for SQL*Net® as a network transport type.

  To use the SQL*Net network transport, you must specify the SQLSRV_XPT_SQLNET argument in the xpttyp field in the associate structure, and you must specify the node_name parameter of the sqlsrv_associate routine as either the SQL*Net Service Name or the SQL*Net Alias.
- Oracle SQL/Services V7.0 is now truly multiversioned on OpenVMS.

  Beginning with Oracle SQL/Services V7.0 on OpenVMS, you can run multiple versions of Oracle SQL/Services server on the same node (V6.0 or V6.1 and V7.0). The Oracle SQL/Services V7.0 client API allows you to specify alternate network ports to enable you to select to which version of a server to connect.
- Support for the Windows 95 client API and the Oracle ODBC Driver for Rdb for the Windows 95 operating system.
- Support for the Solaris client API for the Solaris operating system.
- The MS–DOS large memory model client API is deprecated for Oracle SQL/Services V7.0 and is frozen at the Oracle SQL/Services V6.1 level.
- The ULTRIX for RISC client API is deprecated for Oracle SQL/Services V7.0 and is frozen at the Oracle SQL/Services V6.1 level.

- The SunOS client API is deprecated for Oracle SQL/Services V7.0 and is frozen at the Oracle SQL/Services V6.1 level.
- info_type parameter value SQLSRV_INFO_DB_CLASS of the sqlsrv_get_associate_info routine is deprecated in Oracle SQL/Services V7.0. This parameter value will continue to work for Oracle SQL/Services V7.0; however, Oracle Corporation recommends for V7.0 that you use the new info_type parameter value SQLSRV_INFO_SERVICE_ATTRS. See sqlsrv_get_associate_info for more information.
- info_type parameter value SQLSRV_INFO_SERVICE_ATTRS of the sqlsrv_get_associate_info routine is new in Oracle SQL/Services V7.0. This parameter value replaces the info_type parameter value SQLSRV_INFO_DB_CLASS that is deprecated in Oracle SQL/Services V7.0. See sqlsrv_get_associate_info for more information.
- Support for 32,000-byte buffer sizes.

  The maximum supported buffer size is 32,000 bytes for the read_buffer_size and write_buffer_size parameters in the sqlsrv_associate routine for all transports and platform combinations except NetWare on Windows. Clients negotiate with the V7.0 server for the actual supported maximum buffer size. See sqlsrv_associate for more information.
- A new value, SQLSRV_INFO_BUFFER_SIZE, of the info_type argument of the sqlsrv_get_associate_info routine.

  The value SQLSRV_INFO_BUFFER_SIZE is a new value of the info_type argument of the sqlsrv_get_associate_info routine. This value gets the negotiated buffer size and returns the information as a longword.
- Three ASSOCIATE_STR fields added in V6.1 for internal use only are now available for general use.

  The attach, declare, and appnam fields of the ASSOCIATE_STR can now be used by customers' applications. See Section 7.2 for more information.
- Two new client log field values: SQLSRV_LOG_FLUSH and SQLSRV_LOG_BINARY.

  The SQLSRV_LOG_FLUSH value, when set, flushes pending output to the log file only at the end of each complete association-level, routine-level, and protocol-level log entry and is useful if a client application is terminating abnormally while executing application code.

  The SQLSRV_LOG_BINARY value, when set, dumps memory in structured format if the data contains nonprintable characters.

  See Section 5.1 for more information.

- sqlsrv_execute routine is deprecated in V7.0.

  The routine sqlsrv_execute is deprecated in Oracle SQL/Services V7.0. Oracle Corporation recommends that you code your applications using the sqlsrv_execute_in_out routine. See sqlsrv_execute_in_out for a complete description of the sqlsrv_execute_in_out routine.

Technical changes have been made where necessary to provide technical clarifications, to fix errors of omission, and to make corrections.

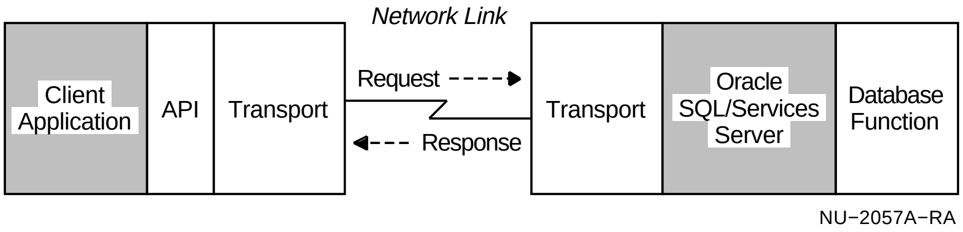
1 Overview

Oracle SQL/Services is a client/server system that enables client applications on PCs and workstations to access data in Oracle Rdb databases on server systems. Oracle SQL/Services follows the client/server model in which:

- The client requests a set of services from the server through an agreed upon interface.
- The server responds by accepting client requests, calling the server function to execute requests, and sending results back to the client.

A simplified view of Oracle SQL/Services is shown in Figure 1–1.

Figure 1–1 Client/Server Model for Oracle SQL/Services



In its implementation of the client/server model, Oracle SQL/Services enables programmers working on any of several computing platforms shown in Table 1–1 to develop client applications that remotely access server databases stored on any operating system supported by the Oracle SQL/Services server using an available network transport.

Table 1–1 Network Transports Supported by Oracle SQL/Services Clients

Clients	DECnet	TCP/IP	SQL*Net	NetWare	AppleTalk
MS Windows	X	X	X	X[1]	–
Windows 95	X	X	–	–	–
Windows NT X86	X	X	–	–	–
Windows NT Alpha	X	X	–	–	–
Macintosh	X	X	–	–	X[2]
Solaris	–	X	–	–	–
Digital UNIX	X	X	X	–	–
OpenVMS Alpha	X	X	X	–	–
OpenVMS VAX	X	X	X	–	–

[1] The NetWare (IPX/SPX) Transport is supported only by servers running on the OpenVMS operating system.

[2] Macintosh clients using the AppleTalk-DECnet Gateway transport communicate with an OpenVMS or Digital UNIX server running DECnet via an AppleTalk-DECnet Gateway node.

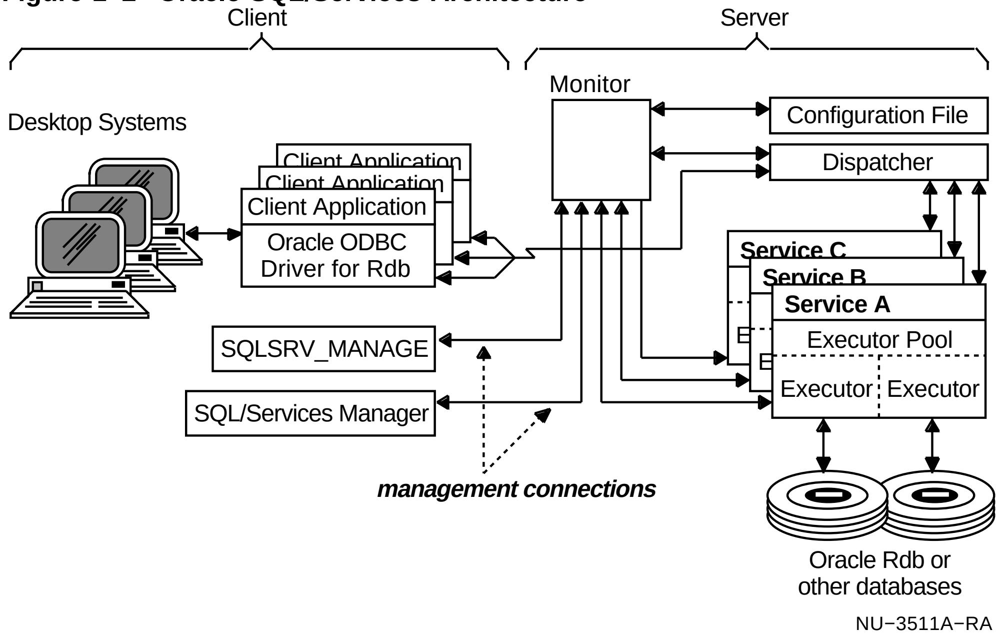
Server databases that client applications can access include either Oracle Rdb databases or other databases supported by Oracle Rdb SQL.

1.1 Introduction to Oracle SQL/Services

Remote application access through Oracle SQL/Services to databases on the server system requires a system configuration similar to the one illustrated in Figure 1–2. Although your system may not exactly mirror the one shown, it must have at least client, network, and server system components.

Section 1.1.1, Section 1.1.2, and Section 1.1.3 briefly describe the client, network, and server system components respectively. Each section identifies the role the component plays in allowing client application access to databases on the server system.

Figure 1–2 Oracle SQL/Services Architecture



1.1.1 Client Components

Client application programs access Oracle SQL/Services on a server node using the Oracle SQL/Services client API. The Oracle SQL/Services client API is a library of callable routines that use layered communications software to communicate with the server node.

- Client API routines

  The Oracle SQL/Services client API routines provide an interface to client applications that is functionally very similar to the dynamic SQL interface. This enables client applications to execute SQL statements against data stored in a database on a server node. The SQL statements can either be defined as string constants in the source code or formulated at run time. The SQL statement syntax accepted by Oracle SQL/Services is identical to that of the dynamic SQL interface of Oracle Rdb.

- Communications software

  Communications software facilitates the transfer of information between the client and server systems. Using a request/response protocol that is virtually transparent to the application, the API accepts client application input, builds Oracle SQL/Services request messages, and transmits them to the server system using DECnet, AppleTalk (through an AppleTalk-DECnet Gateway), Transmission Control Protocol/Internet Protocol (TCP/IP), SQL*Net, or NetWare (IPX/SPX) communications protocol. (See Section 1.1.2 for descriptions of these network components.) Because the Oracle SQL/Services client API provides an interface that is functionally very similar to the dynamic SQL interface, programmers need not understand the communications software to develop Oracle SQL/Services client applications.

Oracle SQL/Services currently supports API software for the client systems described in Table 1–1. See Section 1.2 for more information on supported client platforms.

1.1.2 Network Components

The appropriate client API software can communicate with the Oracle SQL/Services server using DECnet, AppleTalk, TCP/IP, SQL*Net, or NetWare (IPX/SPX) communications software:

- DECnet software

  The DECnet network transport is supported by all Oracle SQL/Services server platforms and by all Oracle SQL/Services client platforms with the exception of Solaris.

- AppleTalk-DECnet software

  The Macintosh Communications Toolbox AppleTalk-DECnet Gateway tool allows a Macintosh system with an AppleTalk network connection to communicate with an Oracle SQL/Services server using DECnet via an AppleTalk-DECnet Gateway node.

- TCP/IP software

  The TCP/IP network transport is supported by all Oracle SQL/Services client and server platforms.

- NetWare (IPX/SPX) software

  The NetWare (IPX/SPX) network transport is supported on OpenVMS servers and MS Windows V3.1 clients only.

- SQL*Net software

  The SQL*Net network transport is supported by all Oracle SQL/Services server platforms and by the MS Windows V3.1, OpenVMS, and Digital UNIX client platforms.

Oracle SQL/Services uses SQL*Net as a network transport to send Oracle SQL/Services protocol messages between Oracle SQL/Services clients and servers. The following additional features are supported with Oracle SQL/Services using SQL*Net:

- Secure Network Services®

  Secure Network Services encrypts and performs security checks on data as it moves across LANs and WANs, preventing any unauthorized user from viewing or tampering with information. Specifically, Secure Network Services provides:

  - Network authentication
  - Tamper-proof data
  - High-speed global data encryption
  - Cross-protocol data security

- Diagnostic tools (tracing and logging)

  Diagnostic tools include Oracle Trace™ and SQL*Net logging.

Regardless of the communications software used, Oracle SQL/Services relieves application programmers of any need to understand networking to develop Oracle SQL/Services applications.

See the *Oracle SQL/Services Release Notes* for network, transport, client, and server operating system version information.

1.1.3 Server System Components

The server system accepts request messages from the application through network transport software, processes the requests against a server system database, and sends response messages back to the waiting application on the client system. For a detailed discussion of the server and its components for the OpenVMS and Digital UNIX platforms, see the *Oracle SQL/Services Server Configuration Guide*.

1.2 Supported Client Platforms

Oracle SQL/Services supports the following client platforms:

- MS Windows V3.1, Windows 95, and Windows NT clients

  The Oracle SQL/Services client API is shipped as a Dynamic Link Library (DLL) on all Windows platforms. You use Microsoft C to develop client applications that you link against the platform-specific DLL to access the Oracle SQL/Services client API. The name of the DLL file for MS Windows

V3.1 clients is sqsapiw.dll; the name of the DLL file for Windows 95, Windows NT X86, and Windows NT Alpha clients is sqsapi32.dll.

All three Windows platforms support the use of an .ini file to customize various aspects of Oracle SQL/Services client API operations including communications, client logging, and so forth. The name of the .ini file for MS Windows V3.1 clients is sqsapiw.ini; the name of the .ini file for Windows 95, Windows NT X86, and Windows NT Alpha clients is sqsapi32.ini. The .ini file that is provided by the installation procedure has all the customizations commented out. You can tailor the operation of the Oracle SQL/Services client API to your specific requirements by reading the directions, then uncommenting and providing appropriate values for the options you need to set.

The Oracle SQL/Services MS Windows V3.1, Windows 95, Windows NT X86, and Windows NT Alpha client API software supports the following network transports:

- DECnet
- TCP/IP
- NetWare (MS Windows V3.1 only)
- SQL*Net (MS Windows V3.1 only)

Client applications on all Windows platforms select the DECnet, TCP/IP, SQL*Net, or NetWare transport using an Oracle SQL/Services client API service or using an .ini file. Specifying a transport in an .ini file overrides a selection made using the Oracle SQL/Services client API service. However, you must use the Oracle SQL/Services client API routine to select the SQL*Net network transport because Oracle SQL/Services requires different arguments for the SQL*Net transport than for other transports. If you are connecting to a server node running multiple versions of Oracle SQL/Services, then you must use an .ini file to select an alternate DECnet, TCP/IP, or NetWare network port if the server you are using does not use the default network ports. If you are using the NetWare transport, there are other options related to NetWare that you may need to set. See the .ini file on your platform for more information on setting Oracle SQL/Services client API options.

- Macintosh clients

  Oracle SQL/Services client API software is available for the Macintosh Programmers Workbench (MPW) development environment from Apple Computer and the THINK C development environment from Symantec. The Oracle SQL/Services MPW client API software is shipped as an MPW object library file. The Oracle SQL/Services THINK C client API software is shipped as two THINK C library files.

The Oracle SQL/Services Macintosh client API software support the following network transports:

- DECnet using the Communications Toolbox DECnet Tool
- AppleTalk using the Communications Toolbox AppleTalk-DECnet Gateway Tool
- TCP/IP using MACtcp

Macintosh client applications select the network transport using an Oracle SQL/Services client API routine or by using the Oracle SQL/Services control panel. Selecting a transport using the Oracle SQL/Services client API overrides the transport selection in the Oracle SQL/Services control panel. If you are connecting to a server node running multiple versions of Oracle SQL/Services, then you must use the control panel to select the transport and to specify an alternate DECnet or TCP/IP network port if the server you are using does not use the default network ports. You may also use the Oracle SQL/Services control panel to select the AppleTalk-DECnet Gateway node when using AppleTalk.

- Solaris client

  The Oracle SQL/Services Solaris client API software is shipped as an object library against which you link your client application programs.

  The Oracle SQL/Services Solaris client API software supports only the TCP/IP network transport. If you are connecting to a server node running multiple versions of Oracle SQL/Services, then you use the SQLSRV_TCPIP_PORT environment variable to select an alternate network port number if the server you are using does not use the default network ports.

- Digital UNIX client

  The Oracle SQL/Services Digital UNIX client API software is shipped as an object library against which you link your client application programs.

  The Oracle SQL/Services Digital UNIX client API software supports the DECnet, TCP/IP, and SQL*Net network transports. If you are connecting to a server node running multiple versions of Oracle SQL/Services and the server you are using does not use the default network ports, then you use the SQLSRV_DECNET_OBJECT environment variable to select an alternate DECnet object or the SQLSRV_TCPIP_PORT environment variable to select an alternate TCP/IP network port number.

- OpenVMS clients

The Oracle SQL/Services OpenVMS VAX and OpenVMS Alpha client API software is shipped as shared images against which you link your client application programs.

The Oracle SQL/Services OpenVMS client API software supports the DECnet, TCP/IP, and SQL*Net network transports. If you are connecting to a server node running multiple versions of Oracle SQL/Services and the server you are using does not use the default network ports, then you use the SQLSRV$DECNET_OBJECT logical name to select an alternate DECnet object or the SQLSRV$TCPIP_PORT logical name to select an alternate TCP/IP network port number.

1.3 Preparing Programmers to Use Oracle SQL/Services

This section describes what application programmers must know to develop applications, and provides a recommended reading path for learning how to develop applications.

1.3.1 What Programmers Must Know to Write Applications

As a programmer creating Oracle SQL/Services applications, you must be familiar with the following:

- C programming language

  Have experience in writing programs in the C programming language. Know how to call Oracle SQL/Services client API routines from C programs to create Oracle SQL/Services applications.

  OpenVMS client applications can be written in any language that supports the OpenVMS Calling Standard.

- Client system environment

  Know how to invoke and use a text editor on your client system to create programming source files. Be able to run your system's C compiler and linker and run the resulting executable image.

- SQL language (and the dynamic SQL interface) concepts

  Have a working knowledge of the SQL language. A conceptual familiarity with the dynamic SQL interface of Oracle Rdb can help you understand the client API routines.

- Oracle SQL/Services API

  Understand how to use the client API routines in your C applications.

1.3.2 Reading Path for Programmers

As a programmer assigned to write client applications, you can become familiar with the process of developing applications using Oracle SQL/Services by reading this guide as follows:

- Chapter 2 helps you to understand the relationship between the dynamic SQL interface and the client API routines, the function of the SQL Communications Area (SQLCA) and the SQL Descriptor Area (SQLDA or SQLDA2) data structures in Oracle SQL/Services, and how to build applications using the Oracle SQL/Services callable API.
- Chapter 3 introduces you to an Oracle SQL/Services sample application that illustrates how to use the Oracle SQL/Services callable client API routines, and includes information on how to compile, link and run the sample application on all the client platforms supported by Oracle SQL/Services.
- Chapter 6 helps you to understand the client API routines that you call from your applications. The chapter provides detailed reference information about all routines in the API callable library.
- Chapter 7 presents detailed reference descriptions of the Oracle SQL/Services data structures.
- Chapter 8 describes the data types used in Oracle SQL/Services.

Other chapters in this guide will support you in your programming as you refine your application development skills.

1.4 Location of Oracle SQL/Services Error Documentation

Programmers developing Oracle SQL/Services API client applications can encounter error messages from a variety of sources:

- Oracle SQL/Services

  When error mnemonics are preceded by SQLSRV_, refer to the sqlsrv.h file and Oracle SQL/Services help for descriptions of errors generated by Oracle SQL/Services client API routines and the Oracle SQL/Services server. Chapter 6 of this guide describes the specific errors that can be returned by each Oracle SQL/Services client API routine.
- SQL

  When error mnemonics are preceded by SQL_, refer to the SQL documentation and SQL help for further error information.

- Oracle Rdb

  When error mnemonics are preceded by SQL_RDBERR_, refer to the *Oracle Rdb7 SQL Reference Manual*, the *Oracle Rdb7 Guide to SQL Programming*, and Oracle Rdb help for pointers to error information.

- Network

  When you receive the primary SQLSRV_NETERR or SQLSRV_HOSTERR errors, look at the network error documentation for the network error referred to in the secondary error status. Refer to the *Oracle SQL/Services Installation Guide* for more information.

1.5 What System Managers Must Know to Support Oracle SQL/Services

If you are the person responsible for managing Oracle SQL/Services at your site, see the *Oracle SQL/Services Installation Guide* and the *Oracle SQL/Services Server Configuration Guide*.

Information about installing the client API software for all interfaces supported by Oracle SQL/Services is not included in this document. Refer to the *Oracle SQL/Services Installation Guide* for instructions on installing the OpenVMS clients and to the readme files provided on the Oracle Rdb Client CD–ROM for installing all other clients described in Table 1–1.

2

Developing Oracle SQL/Services Applications

This chapter describes a number of topics programmers must understand before writing client applications. Topics covered in this chapter include:

- A description of the dynamic SQL interface for Oracle Rdb

  The Oracle SQL/Services client API routines that programmers use in client applications to access the dynamic SQL interface on the server system correspond closely to the dynamic SQL interface statements. An understanding of the dynamic SQL interface can help programmers understand the way the client API routines work. See Sections 2.1 to 2.3.

- An overview of Oracle SQL/Services client API routines

  Programmers include in their applications calls to the Oracle SQL/Services client API routines to access Oracle SQL/Services functions on the server system. Client applications link against the Oracle SQL/Services client API library, DLL, or shared image to access these routines. See Section 2.4.

- An overview of Oracle SQL/Services data structures

  The Oracle SQL/Services client API routines use a set of data structures that allow two-way communication between applications on the client system and SQL on the server system. See Section 2.5.

- A recommended approach to developing Oracle SQL/Services applications

  Oracle Corporation recommends that you let Oracle SQL/Services allocate memory for SQLCA, SQLDA, and SQLDA2 data structures and that you use functional interface routines to access these data structures. This approach is a requirement for developing applications designed to run on the Macintosh platform. See Section 2.6.

- Steps for building Oracle SQL/Services application programs

  Programmers must compile and link their applications to create an executable image that can access Oracle SQL/Services. The steps to link an application program differ from one client system to another and are thus provided for each client system. See Section 2.7.

If you are already familiar with the dynamic SQL interface, you may want to skip to Section 2.4, which describes the structures used by Oracle SQL/Services client API routines.

2.1 Introduction to the Dynamic SQL Interface of Oracle Rdb

The **dynamic SQL** interface of Oracle Rdb allows application programs to formulate and execute SQL statements at run time. It consists of:

- Dynamic SQL statements

  A set of SQL statements with which you can write applications using either the SQL precompiler or the SQL module processor

- Data structures

  A set of data structures that provides a way for the dynamic SQL interface and application programs to exchange data and metadata

Applications that use the dynamic SQL interface might, for example, translate interactive user input into SQL statements, or open, read, and execute files containing SQL statements. The Oracle SQL/Services executor is itself a dynamic SQL interface application.

For more detailed information on the dynamic SQL interface of Oracle Rdb, see the *Oracle Rdb7 Guide to SQL Programming* and the *Oracle Rdb7 SQL Reference Manual*.

2.2 Overview of Dynamic SQL Interface Statements

The dynamic SQL interface statements are summarized in Section 2.2.1 and Section 2.2.2, which group the statements according to function. For each dynamic SQL interface statement, there is an Oracle SQL/Services client API routine that performs the same function. Some client API routines, like sqlsrv_prepare, combine the functions of two dynamic SQL interface statements.

2.2.1 Execution Statements

Execution statements prepare and execute SQL statements and release prepared SQL statement resources.

- PREPARE

  Compiles the SQL statement, checking it for errors, and returns a handle to the prepared statement. The handle is subsequently used to reference the prepared statement.

- DESCRIBE

  Stores the number and metadata information of any select list items or parameter markers in an SQLDA structure.

- EXECUTE

  Executes a previously prepared SQL statement that is not a SELECT statement.

- EXECUTE IMMEDIATE

  Prepares and executes in one step any SQL statement (other than SELECT) that does not contain parameter markers or select list items.

- RELEASE

  Releases all resources used by a prepared SQL statement.

Except for the DESCRIBE statement, each of these dynamic SQL statements has an equivalent Oracle SQL/Services routine. In Oracle SQL/Services, the DESCRIBE and PREPARE statements are combined in a single routine, as shown in Table 2–2.

2.2.2 Result Table Statements

Result table statements allow your program to declare a cursor, open a cursor, fetch data from an open cursor, and close an open cursor.

- DECLARE CURSOR

  Declares a cursor for a prepared SELECT statement.

- OPEN

  Opens a cursor declared for a prepared SELECT statement.

- FETCH

  Retrieves values from a cursor declared for a prepared SELECT statement.

- CLOSE

  Closes a cursor.

2.3 Using the Dynamic SQL Interface of Oracle Rdb

Note

The following general discussion is relevant only to the dynamic SQL interface. Some of the functionality described in this section may not be directly accessible to an Oracle SQL/Services client application.

You can execute the simplest SQL statements that neither accept variable data values from nor return data values to your application using the EXECUTE IMMEDIATE dynamic SQL statement. If you use EXECUTE IMMEDIATE to execute a statement, SQL automatically prepares, executes, and releases the statement for you. However, if you need to execute the same SQL statement more than once, using EXECUTE IMMEDIATE is inefficient because SQL must prepare and release the statement each time it is executed. In this situation, it is more efficient for your application to prepare the statement, execute it as many times as necessary, and release it only when it is no longer needed.

More complex SQL statements can accept variable data values from or return data values to your application. Your application provides variable data values to SQL statements as parameter markers, using a question mark character (?) to identify each parameter marker. A SELECT statement will return a select list item for each column named in the select list clause. In addition, you also identify the data values returned by singleton-SELECT, UPDATE . . . RETURNING, and CALL statements using a question mark character (?) for each returned data value.

To process more complex SQL statements with parameter markers or select list items, and to improve the efficiency of your application when processing SQL statements that are used multiple times, you first use PREPARE to dynamically compile the statement. You then optionally use DESCRIBE to obtain the metadata for any parameter markers or select list items. You use the EXECUTE statement to process executable SQL statements, such as INSERT, UPDATE, DELETE, singleton-SELECT, CALL, and compound statements. To process a result table formed by a SELECT statement, you first use DECLARE CURSOR and OPEN to declare and open a cursor. You then use FETCH to retrieve rows from the result table. Finally, you use CLOSE to close the cursor at the end. When a statement is no longer needed, you free the resources used by the prepared statement using the RELEASE statement.

Section 2.3.1 describes how to use dynamic SQL operations to process statements that contain parameter markers. Section 2.3.2 describes how to access the data returned by SELECT statements. Section 2.3.3 describes how to handle statements about which the program has no prior information.

Table 2–1 lists the major SQL statements that can be processed using dynamic SQL. However, certain SQL statements cannot be processed using dynamic SQL. This includes all the SQL statements listed in Table 2–2 including those that comprise the dynamic SQL interface itself. Furthermore, statements and commands such as SHOW that are processed only by the interactive SQL utility cannot be processed using the dynamic SQL interface.

Table 2–1 SQL Statements That Can Be Processed Using Dynamic SQL Operations

Statement	Associated Dynamic SQL Statements
SELECT	PREPARE, Extended Dynamic DECLARE CURSOR, DESCRIBE (optional), OPEN, FETCH, CLOSE, RELEASE
INSERT, UPDATE, DELETE, CALL, Singleton-SELECT, ATTACH, CONNECT, SET CONNECT, DISCONNECT	PREPARE, DESCRIBE (optional), EXECUTE and RELEASE, or EXECUTE IMMEDIATE (if no parameter markers or select list items)
CREATE, ALTER, DROP, DECLARE TRANSACTION, SET TRANSACTION, COMMIT, ROLLBACK, GRANT, REVOKE, COMMENT ON	PREPARE, EXECUTE and RELEASE, or EXECUTE IMMEDIATE

Table 2–2 SQL Statements That Cannot Be Processed Using Dynamic SQL Operations

SQL Statement	Related Oracle SQL/Services Routine
BEGIN DECLARE	none
CLOSE	sqlsrv_close_cursor
DECLARE ALIAS	none
DECLARE CURSOR	sqlsrv_declare_cursor
DECLARE STATEMENT	none

(continued on next page)

Table 2–2 (Cont.) SQL Statements That Cannot Be Processed Using Dynamic SQL Operations

SQL Statement	Related Oracle SQL/Services Routine
DECLARE TABLE	none
DESCRIBE	sqlsrv_prepare (implicit in)
END DECLARE	none
EXECUTE	sqlsrv_execute_in_out
EXECUTE IMMEDIATE	sqlsrv_execute_immediate
FETCH	sqlsrv_fetch
INCLUDE	none
OPEN	sqlsrv_open_cursor
PREPARE	sqlsrv_prepare
RELEASE	sqlsrv_release_statement
WHENEVER	none

2.3.1 Parameter Markers

Parameter markers represent variables that can be processed using dynamic SQL operations with SQL SELECT, INSERT, UPDATE, DELETE, CALL, Singleton-SELECT, ATTACH, CONNECT, SET CONNECT, and DISCONNECT statements. Question marks (?) embedded in the statement string denote parameters that are to be replaced when the statement is processed using the dynamic SQL interface. An example of an SQL statement with parameter markers is:

```
INSERT INTO EMPLOYEES
      (EMPLOYEE_ID, FIRST_NAME, LAST_NAME, CITY)
      VALUES ( ?, ?, ?, ? );
```

The mechanism for mapping parameter markers to variables in application programs is a data structure called the SQLDA or SQLDA2 (see Section 2.3.4 and Section 7.5). The DESCRIBE statement writes information about parameter markers into an SQLDA or SQLDA2 structure. Your program examines the SQLDA or SQLDA2 structure, allocates a data variable and an indicator variable for each parameter marker, obtains values for each parameter marker, and stores the values in the SQLDA or SQLDA2 data variables before processing the SQL statement using the dynamic SQL interface.

2.3.2 Select List Items

Programs that process SELECT statements using dynamic SQL operations must declare a cursor to receive the result table, and must allocate memory for each select list item in the SELECT statement. After the cursor is opened, FETCH statements return values for rows of the result table.

INSERT . . . RETURNING, UPDATE . . . RETURNING, CALL, compound statements and singleton-SELECT statements are executable statements that are processed using the EXECUTE dynamic SQL statement that can return information in a select list SQLDA. For example,

```
UPDATE EMPLOYEES SET SALARY=SALARY+? WHERE BADGE=? RETURNING SALARY INTO ?;
```

As with parameter markers, the mechanism for mapping select list items to host variables is a data structure called the SQLDA or SQLDA2 (see Section 2.3.4 and Section 7.5). The DESCRIBE statement writes select list information into the SQLDA or SQLDA2.

If the SQL statement contains parameter markers in addition to select list items, the program must also set up host variables for the parameter markers and assign values to them.

2.3.3 Unknown Statements

It is possible to process SQL statements using the dynamic SQL interface about which the program has no prior information. Such statements may contain parameter markers or select list items or both. The program can use the DESCRIBE statement to obtain an SQLDA or SQLDA2 structure containing information about the numbers and data types of select list items and parameter markers. Then the program allocates data and indicator variables as appropriate and writes the addresses of those variables into the SQLDA or SQLDA2 structures before executing the statement.

2.3.4 SQL Descriptor Area (SQLDA or SQLDA2)

The SQL Descriptor Area **(SQLDA)** or Extended SQL Descriptor Area **(SQLDA2)** is a data structure that enables programs to communicate with SQL about parameter markers and select list items.

Oracle Rdb SQL provides an extended version of the SQLDA structure, called the SQLDA2, which supports additional fields and field sizes. Oracle SQL/Services supports this SQLDA2 structure. For more information about the SQLDA2 data structure and its use with the SQL interface of Oracle Rdb, refer to Section 7.5 and to the appendix of the *Oracle Rdb7 SQL Reference Manual*.

When SQL processes a DESCRIBE statement, it writes information about select list items (for a DESCRIBE . . . SELECT LIST statement) or parameter markers (for a DESCRIBE . . . MARKERS statement) of a prepared statement into an SQLDA or SQLDA2.

The host language program examines the SQLDA or SQLDA2 to determine how many select list items or parameter markers are present and the data type of each. The program must provide memory for data and indicator variables for each parameter marker or select list item, and write the address of each memory location into the SQLDA or SQLDA2.

For parameter markers, the program writes values into the SQLDA or SQLDA2 before processing the SQL statement using dynamic SQL operations. For select list items, the program reads the data written into the SQLDA or SQLDA2 by subsequent FETCH statements.

The *Oracle Rdb7 SQL Reference Manual* contains an appendix on the SQLDA and SQLDA2 and a section on the DESCRIBE statement that discusses the MARKERS and SELECT LIST clauses of the DESCRIBE statement in more detail.

2.3.5 SQL Communications Area (SQLCA)

The SQL Communications Area **(SQLCA)** is a data structure that SQL uses to provide information about the execution of SQL statements to application programs. SQL updates the contents of the SQLCA after completion of every executable SQL statement. Fields of interest in the SQLCA are the SQLCODE field and several elements of the SQLERRD array.

The SQLCODE field contains the completion status of every SQL request.

Both SQL and Oracle SQL/Services may store information in one or more elements of the SQLERRD array to provide additional details about the execution of a SQL statement. For example, SQL stores the statement type in the SQLERRD array following a PREPARE request; while Oracle SQL/Services stores additional network error information in the SQLERRD array if an associate fails due to a network error.

See Section 7.4 for a description of the other values of the SQLERRD array. Section 7.3 describes the SQLCA in detail. In addition, the *Oracle Rdb7 SQL Reference Manual* contains an appendix on the SQLCA.

2.4 Overview of Client API Routines

The Oracle SQL/Services client application programming interface (API) is a set of callable routines that client appplications use to access Oracle SQL/Services functions. The client API routines are grouped according to function and summarized in Section 2.4.1 through Section 2.4.5.

2.4.1 Association Routines

Association routines create and terminate client/server associations and control the association environment. These routines are:

- sqlsrv_abort

  Terminates a client/server association. Disconnects from the server and releases all client resources related to the association.

- sqlsrv_associate

  Creates a client/server association. Makes the remote connection to the server process and negotiates association characteristics and attributes.

- sqlsrv_get_associate_info

  Gets association information.

- sqlsrv_release

  Terminates a client/server association in an orderly fashion. Sends a message to the server requesting termination of the association, disconnects the network link, and releases all client resources related to the association.

2.4.2 SQL Statement Routines

SQL statement routines prepare and execute SQL statements, and release prepared SQL statement resources. These routines map directly to the dynamic SQL interface. These routines are:

- sqlsrv_prepare

  Prepares a dynamic SQL statement. It returns a statement identifier and SQLDA or SQLDA2 metadata information. This routine maps to the dynamic SQL interface PREPARE and DESCRIBE statements.

- sqlsrv_execute_in_out

  Executes a prepared SQL statement. This routine maps to the dynamic SQL interface EXECUTE statement.

- sqlsrv_execute_immediate

  Prepares and executes an SQL statement. This routine cannot be used if the SQL statement contains parameter markers or select list items. This routine maps to the dynamic SQL interface EXECUTE IMMEDIATE statement.

- sqlsrv_release_statement

  Releases client and server statement resources associated with a prepared statement. This routine maps to the dynamic SQL interface RELEASE statement.

2.4.3 Result Table Routines

Result table routines allow the caller to fetch data from the server by providing calls to open a cursor, fetch from an open cursor, and close an open cursor. These routines are:

- sqlsrv_declare_cursor

  Declares the type and mode of an extended dynamic cursor. Note that the cursor is actually declared at the server when sqlsrv_open_cursor is called the first time for a specific cursor name. If you do not call the sqlsrv_declare_cursor routine for a particular cursor name before calling sqlsrv_open_cursor, Oracle SQL/Services implicitly declares the cursor as type table and mode update.

  This routine conceptually maps to the dynamic SQL interface DECLARE CURSOR statement.

- sqlsrv_open_cursor

  Opens a cursor by associating a cursor name with a prepared statement identifier. The cursor name is used in each reference to the cursor. The sqlsrv_open_cursor routine also declares the extended dynamic cursor at the server the first time it is called for a specific cursor name.

  This routine conceptually maps to the dynamic SQL interface OPEN statement.

- sqlsrv_fetch

  Fetches one row of data from an open cursor.

  This routine maps to the dynamic SQL interface FETCH statement.

- sqlsrv_fetch_many

  Requests that multiple rows of data be fetched and transmitted to the client, which frequently reduces the number of network messages.

  This routine has no equivalent dynamic SQL interface statement. Rather, it controls the way the server sends row data back to the client after it has been retrieved by the server using the dynamic SQL interface FETCH statement.

- sqlsrv_close_cursor

  Closes an open cursor.

  This routine maps to the dynamic SQL interface CLOSE statement.

2.4.4 Utility Routines

Utility routines provide miscellaneous services to the caller. These routines are:

- sqlsrv_allocate_sqlda_data or sqlsrv_allocate_sqlda2_data

  Allocates memory for the SQLDA or SQLDA2 data buffer and indicator variable fields.

- sqlsrv_free_sqlda_data or sqlsrv_free_sqlda2_data

  Frees memory for the SQLDA or SQLDA2 data buffer and indicator variable fields.

- sqlsrv_set_option

  Sets the option that determines whether an SQLDA or SQLDA2 is used.

2.4.5 Functional Interface Routines

The functional interface routines provide access to data and metadata stored in the SQLCA, SQLDA, and SQLDA2 structures. These routines replace the need for making direct references to structure fields in client applications. These routines are:

- sqlsrv_sqlca_error

  Returns from the SQLCA structure the error codes for the last statement executed.

- sqlsrv_sqlca_error_text

  Returns from the SQLCA structure the error text for the last statement executed.

- sqlsrv_sqlca_count

  Returns from the SQLCA the number of rows processed by a statement and replaces direct access to the SQLCA.SQLERRD[2] field.

- sqlsrv_sqlca_sqlerrd

  Returns to your application the contents of the entire SQLCA.SQLERRD array which includes, for example, optimizer information for a table cursor, and number of segments, maximum segment length, and so forth for a list cursor, following a successful call to sqlsrv_open_cursor.

- sqlsrv_sqlda_sqld or sqlsrv_sqlda2_sqld

  Returns the number of parameter markers or select list items in the SQLDA or SQLDA2 and replaces direct access to the SQLD field in an SQLDA or SQLDA2.

- sqlsrv_sqlda_column_name or sqlsrv_sqlda2_column_name

  Copies the column name for a particular column from the SQLDA or SQLDA2 into the variable passed in this call.

- sqlsrv_sqlda_column_type or sqlsrv_sqlda2_column_type

  Returns from the SQLDA or SQLDA2 information about the data type of a column.

- sqlsrv_sqlda_bind_data or sqlsrv_sqlda2_bind_data

  Allows programs to allocate their own storage for data and indicator variables in an SQLDA or SQLDA2.

- sqlsrv_sqlda_unbind_sqlda or sqlsrv_sqlda2_unbind_sqlda2

  Releases all variables bound with the sqlsrv_sqlda_bind_data or sqlsrv_sqlda2_bind_data routine.

- sqlsrv_sqlda_ref_data or sqlsrv_sqlda2_ref_data

  Returns from the SQLDA or SQLDA2 the type and length and addresses of the data and indicator variables for a column.

- sqlsrv_sqlda_unref_data or sqlsrv_sqlda2_unref_data

  Frees resources tied up by the sqlsrv_sqlda_ref_data or sqlsrv_sqlda2_ref_data routine.

- sqlsrv_sqlda_get_data or sqlsrv_sqlda2_get_data

  Copies data and indicator values from the SQLDA or SQLDA2 to program variables and provides access to SQLDA or SQLDA2 information for languages that do not support explicit type coercion.

- sqlsrv_sqlda_set_data or sqlsrv_sqlda2_set_data

Copies data and indicator values from program variables into the SQLDA or SQLDA2.

- sqlsrv_sqlda_set_sqllen or sqlsrv_sqlda2_set_sqllen

  Sets the length of a column of type SQLSRV_ASCII_STRING, SQLSRV_VARCHAR, and SQLSRV_VARBYTE by setting the SQLLEN field in an SQLDA or SQLDA2. The sqlsrv_sqlda2_set_sqllen also sets the SQLOCTET_LEN in an SQLDA2.

- sqlsrv_sqlda2_char_set_info

  Returns SQL character set information from the SQLDA2.

2.5 Overview of Data Structures

Oracle SQL/Services uses data structures to communicate with the client application. The client API routines use the following data structures:

- ASSOCIATE_STR

  This structure is passed as a parameter to sqlsrv_associate to set the characteristic of an association. The sqlsrv_associate routine opens the communications link between client and server and creates an association. For more information, see Section 7.2.

- SQLCA

  The SQLCA (SQL Communications Area) is used to store error messages and SQL statement information returned by Oracle SQL/Services. When a client API routine returns a nonzero value indicating that an error occurred, the SQLCA contains additional error information. For more information, see Section 7.3.

- SQLDA or SQLDA2

  The SQLDA (SQL Descriptor Area) or SQLDA2 (Extended SQL Descriptor Area) is used to exchange database metadata and data for parameter markers (input) and select list items (output). The Oracle SQL/Services SQLDA or SQLDA2 is identical to that used by the dynamic SQL interface for Oracle Rdb. For more information, see Section 2.3.4 and Section 7.5.

2.6 Developing Applications with the Functional Interface Routines

When designing an application, you must decide how to allocate memory for SQLCA, SQLDA, and SQLDA2 data structures and how to access these data structures.

Oracle Corporation recommends that you let Oracle SQL/Services allocate memory for SQLCA, SQLDA, and SQLDA2 data structures. Note that you must let Oracle SQL/Services allocate memory for applications designed to run on the Macintosh platform. To let Oracle SQL/Services allocate memory for the SQLCA data structure, specify a NULL pointer in the call to sqlsrv_associate. To let Oracle SQL/Services allocate memory for SQLDA and SQLDA2 data structures, specify NULL SQLDA_ID pointers in the call to sqlsrv_prepare. Note that you can direct Oracle SQL/Services to use application-specific memory allocation and deallocation routines by specifying their addresses in the associate data structure (ASSOCIATE_STR) that you pass to sqlsrv_associate. Alternatively, you can allocate memory for SQLCA, SQLDA, and SQLDA2 data structures prior to calling sqlsrv_associate and sqlsrv_prepare on all client platforms except for the Macintosh.

The Oracle SQL/Services client API provides a set of functional interface routines that allow indirect access to the SQLCA, SQLDA, and SQLDA2 data structures. Oracle Corporation recommends that you use the functional interface routines to access the SQLCA, SQLDA, and SQLDA2 data structures to facilitate portability across all supported client platforms. Note that you must use the functional interface routines to develop applications designed to run on the Macintosh platform. See Section 2.4.5 for a complete list of the functional interface routines and a brief description of each routine. Alternatively, you can directly access the SQLCA, SQLDA, and SQLDA2 data structures on all client platforms except for the Macintosh.

2.7 Building Oracle SQL/Services Application Programs

The process of building Oracle SQL/Services application programs consists of these steps:

1. Compile your code using the following *#include* compiler directive:

   ```
   #include <sqlsrv.h>     /* Typedefs, function prototypes, error literals*/
   ```

   If your application accesses the SQLCA, SQLDA, or SQLDA2 structures directly, also include the sqlsrvca.h or sqlsrvda.h header files as follows. These header files are not provided on the Macintosh platform.

```
#include <sqlsrvca.h> /*SQLCA structure */
#include <sqlsrvda.h> /*SQLDA and SQLDA2 structures */
```

On most operating systems, include files are kept in a standard location, indicated in C by placing angle brackets around the name of the file. If these directives do not work on your system, ask the person who installed the Oracle SQL/Services API where the include files are located.

Note

Some C compilers have a problem with %S and %D when printing error messages (for example, %SQLSRV and %DBS).

To avoid this problem, Oracle Corporation recommends that you use either a printf or puts statement when printing Oracle SQL/Services error messages:

```
printf ("%s", message);
```

or

```
puts (message);
```

2. Link your object module with the Oracle SQL/Services client API. Linking procedures are system dependent and are thus discussed separately in the following sections.

   Linking procedures can also depend on the network transport you want to use with Oracle SQL/Services and the specific client.

2.7.1 Building Applications on the OpenVMS Operating System

The OpenVMS include files are installed in SYS$LIBRARY.

To link your program, enter the following command:

```
$ LINK object.obj,SYS$LIBRARY:SQLSRV$API/OPT
```

Replace *object* with the name of your object module.

2.7.2 Building Applications on the MS Windows Operating System

This section describes how to build and run 16-bit applications for MS Windows V3.1 and 32-bit applications for Windows 95, Windows NT X86, or Windows NT Alpha.

2.7.2.1 Building 16-Bit Applications on MS Windows V3.1

The Oracle SQL/Services client API for Windows V3.1 is supplied in the form of a Dynamic Link Library (DLL) called sqsapiw.dll, together with a library file called sqsapiw.lib. Review your Windows V3.1 documentation for information about creating applications that link against a DLL. If you use sqsdynw.mak as a template, you will need to customize it to your application's particular requirements.

If you plan to use the NetWare IPX/SPX network transport, the nwipxspx.dll and nwnetapi.dll files must also be installed on your system. These files may be obtained from the Oracle Rdb Client Kits CD–ROM or the NetWare C Interface SDK.

See the *Oracle SQL/Services Release Notes* for a complete list of software products and their versions that are required to support different network transports.

2.7.2.2 Building 32-Bit Applications for Windows 95, Windows NT X86, or Windows NT Alpha

The Oracle SQL/Services client API for Windows 95, Windows NT X86, or Windows NT Alpha is supplied in the form of a Dynamic Link Library (DLL) called sqsapi32.dll, together with a library file called sqsapi32.lib. Review your Windows documentation for information about creating applications that link against a DLL. If you use sqsdyn32.mak as a template, you will need to customize it to your application's particular requirements.

See the *Oracle SQL/Services Release Notes* for a complete list of software products and their versions that are required to support different network transports.

If you want to call Oracle SQL/Services using threads on Windows NT, you must be aware of the following:

- Oracle SQL/Services synchronizes calls to the Oracle SQL/Services client API routines between threads. That is, only one Oracle SQL/Services call may be active per associate at a time. All subsequent concurrent calls for an association stall until all previous calls complete.
- The error and error messages returned into the SQLCA data structure should *not* be accessed or manipulated directly by the application programmer. This structure will contain the message returned by the last thread that accessed it. Therefore, an error received in one thread may be overwritten by another thread. This may cause the application program to receive the wrong error and associated messages for the thread that

initially received the error. To receive the correct error and messages, use the following Oracle SQL/Services routines:

- sqlsrv_sqlca_error
- sqlsrv_sqlca_error_text
- sqlsrv_sqlca_sqlerrd

2.7.3 Building Applications on the Digital UNIX Operating System

The Digital UNIX include files are installed in the /usr/include directory.

By default, the Digital UNIX C compiler compiles and links your program in one command, including support for both DECnet and TCP/IP. For example:

```
% cc file -lsqs -lots -ldnet -o name
```

Replace *file* with the name of your source file and *name* with the name that you want for the executable file. If your application uses the DECnet transport, include the optional –ldnet argument as shown; otherwise, replace –ldnet with –ldnet–stub.

You may find it useful to examine the makefile that builds the Digital UNIX API Installation Verification Procedure (sqsivpu.mak) and the makefile that builds the sample application, sqsdynu.mak (see Section 3.2.4).

2.7.4 Building Applications on the Macintosh Operating System

The following sections describe how to build Oracle SQL/Services applications on a Macintosh system, either in the Macintosh Programmer's Workshop (MPW) or in the THINK C programming environment.

2.7.4.1 Building Applications on the Macintosh Operating System for MPW

The Macintosh include file sqlsrv.h is installed in the {CLibraries} directory.

To link your program under MPW, enter the following command:

```
Link {Linkoptions} -w -t APPL -c 'creator' ~d~
        "object" ~d~
        "{Libraries}"Runtime.o" ~d~
        "{Libraries}"Interface.o ~d~
        "{CLibraries}"StdCLib.o ~d~
        "{CLibraries}"CSANELib.o ~d~
        "{CLibraries}"Math.o ~d~
        "{Libraries}"CommToolbox.o ~d~
        "{CLibraries}"sqlsrv.o ~d~
        -o "application"
```

The ~d~ symbol refers to the Apple delta character (Option-d). Replace *object* with the name of your object module and *application* with the name of the application you want to create. Replace *creator* with the creator name.

You might find it useful to examine the makefile provided for building the sample application in the MPW environment. However, note that the sample application is linked as an MPW tool and, therefore, uses object libraries in addition to those previously described. The sample application makefile is located in the MPW:Examples:SQLSRV Examples folder. For more information, see Section 3.2.5.1.

2.7.4.2 Building Applications on the Macintosh Operating System for THINK C

The include file sqlsrv.h is normally installed in the Sym C++ for Mac: Macintosh Libraries:THINK #includes folder.

You can build your applications on your Macintosh system for THINK C by following these steps:

1. Launch the THINK C Project Manager.
2. Create a TPM ANSI project to build the application.
3. If necessary, remove main.c by using the remove option from the Source menu.
4. Use the ADD option from the Source menu to add the following files to the project:
   - Your C program source files
   - The sqlsrv1, sqlsrv2, CommToolbox, MacTraps, and MacTraps2 libraries from the Sym C++ for Mac Folder:Macintosh Libraries:68K Libraries folder

   Note

   You must create separate segments for the ANSI, sqlsrv1, and sqlsrv2 libraries (one segment for each library), due to THINK C program size limitations. Otherwise, you will receive this error message: "Code segment too big." Refer to the THINK C documentation for information about creating program segments.

5. Select the Build the Application entry from the Project menu to build the project.

2.7.5 Building Applications on the Solaris Operating Systems

The Solaris include files are installed in the /usr/include directory.

By default, the Solaris C compiler compiles and links your program in one command. For example:

```
% cc file -lsqs -o name
% chmod +x name
```

Replace *file* with the name of your source file and *name* with the name that you want for the executable file.

You may find it useful to examine the makefile that builds the Solaris API Installation Verification Procedure (sqsivpu.mak) and the makefile that builds the sample application, sqsdynu.mak (see Section 3.2.6).

3
Sample Application Guidelines

This chapter guides you through the Oracle SQL/Services sample application.

3.1 Sample Application

Sections 3.1, 3.2, and 3.3 describe a sample interactive application that accepts dynamic SQL statements and processes them using the Oracle SQL/Services client API. The sample application consists of two or three modules, depending on your client platform:

- A driver module named sqsdrv.c (on all Windows platforms), sqsdrvu.c (on all UNIX platforms) or sqlsrv$driver.c (on all OpenVMS and Macintosh platforms). This module accepts dynamic SQL statements from the user and calls the dynamic SQL processing module to process the statements. It is described in Section 3.4.
- A dynamic SQL processing module named sqsdyn.c (on all Windows platforms), sqsdynu.c (on all UNIX platforms) or sqlsrv$dynamic.c (on all OpenVMS and Macintosh platforms). This module accepts dynamic SQL statements from the driver module and calls Oracle SQL/Services client API routines to process the statements. It is described in Section 3.5.
- An I/O module named winivp.c for Windows platforms only. This module calls Windows services to implement a basic Windows I/O interface and is not described in this chapter.

The sample application is able to process any dynamic SQL statement, including executable statements such as INSERT, UPDATE, DELETE, singleton-SELECT, and CALL statements, as well as SELECT statements. To process a statement entered by the user, the sample application first prepares the statement. If a statement contains parameter markers, the sample application then prompts the user for parameter marker values. To process an executable statement, the sample application executes the statement, then displays any results that the statement might produce. To process a SELECT statement, the sample application declares and opens a cursor, fetches and displays rows from the result table, then closes the cursor when all rows have been fetched. Finally, the sample application releases the statement to free the resources held by the prepared statement.

In some respects, the Oracle SQL/Services sample application resembles a limited, portable implementation of the Oracle Rdb interactive SQL application. Like interactive SQL, the driver module recognizes the semicolon (;) as an SQL statement terminator and thus accepts multiline statements. However, unlike interactive SQL, it does not parse the SQL statements entered by the user and thus cannot handle compound statements or the definition of stored procedures. Input lines beginning with an exclamation point (!) are considered comments and are not executed.

3.2 Building the Sample Application

This section describes how to build the sample application on the client platforms supported by Oracle SQL/Services.

3.2.1 Building the Sample Application on the OpenVMS Operating System

The source code for the sample application is available on line in a directory under SYS$EXAMPLES. To copy, compile, link, and run the sample application, enter the following commands:

```
$ copy sys$common:[syshlp.examples.sqlsrv]sqlsrv$*.c *
$ cc sqlsrv$driver,sqlsrv$dynamic
$ link/exe=sqlsrv$dynamic sqlsrv$driver,sqlsrv$dynamic,-
_$ sys$library:sqlsrv$api/opt
$ run sqlsrv$dynamic
```

Note

If Oracle SQL/Services is installed multiversion, copy the sample application files from SYS$COMMON:[SYSHLP.EXAMPLES.SQLSRV70].

3.2.2 Building the Sample Application on MS Windows V3.1

An executable form of the sample application is supplied when you install the Oracle SQL/Services client kit. This executable program was built using the default settings and switches in sqsdynw.mak, and so it might not be suitable for all environments and transports. The executable is named sqsdynkw.exe, where the k indicates the executable was provided with the kit, so it will not be overwritten if you rebuild the sample application locally.

The source files for the sample application are supplied in the directory where you installed the Oracle SQL/Services client kit. The sqsdynw.mak file uses the Microsoft C compiler to create an executable named sqsdynw.exe. Review sqsdynw.mak as a sample guide and for information on default settings and switches.

Use the following commands to build the sample application from the MS–DOS prompt. Select the appropriate NMAKE command depending on whether or not you want to build a debuggable executable.

```
> cd \[sql/services-install-dir]    | Oracle SQL/Services installation directory
> nmake -a -f sqsdynw.mak          | To build a nodebug executable, or
> nmake -a -f sqsdynw.mak debug=1  | to build a debuggable executable
> win sqsdynw.exe                  | Invoke sample after successful build
```

3.2.3 Building the Sample Application on Windows NT X86, Windows NT Alpha, or Windows 95

An executable form of the sample application is supplied when you install the Oracle SQL/Services client kit. This executable program was built using the default settings and switches in sqsdyn32.mak, and so it might not be suitable for all environments and transports. The executable is named sqsdyn32.exe; you may wish to copy or rename this file if you rebuild the sample application locally.

The source files for the sample application are supplied in the directory where you installed the Oracle SQL/Services client kit. The sqsdyn32.mak file uses the Microsoft C compiler to create an executable named sqsdyn32.exe. Review sqsdy32.mak as a sample guide and for information on default settings and switches.

Use the following commands to build the sample application from the MS–DOS prompt. Select the appropriate NMAKE command depending on whether or not you want to build a debuggable executable.

```
> cd \[sql/services-install-dir]    | Oracle SQL/Services installation directory
> nmake -a -f sqsdyn32.mak         | To build a nodebug executable, or
> nmake -a -f sqsdyn32.mak debug=1 | to build a debuggable executable
> sqsdyn32                         | Invoke sample after successful build
```

3.2.4 Building the Sample Application on the Digital UNIX Operating System

If DECnet *is* available on your system, you can build the Digital UNIX sample application by issuing the following command:

```
make -f sqsosfsample
```

If DECnet *is not* available on your system, you can build the Digital UNIX sample application by issuing the following command:

```
make -f sqsosfsample "DNETLIB= -ldnet_stub"
```

See Section 2.7.3 for information on building applications on Digital UNIX systems.

3.2.5 Building the Sample Application on the Macintosh Operating System

The following sections describe how to build and run the Oracle SQL/Services sample application on the Macintosh, either in the Macintosh Programmer's Workshop (MPW) or in the THINK C programming environment.

3.2.5.1 Building the Sample Application in the MPW Programming Environment

You can build the sample application on your Macintosh system by performing the following tasks:

1. Launch the Macintosh Programmer's Workshop (MPW).
2. Set your default directory by typing the following directory specification and pressing Enter:

   ```
   directory {MPW}Examples:"Sqlsrv Examples"
   ```

   Alternatively, on MPW systems that have been customized, you can pull down the Directory menu and select SQLSRV Examples.
3. Build the SQLSRV$DYNAMIC sample application by typing the following command and pressing Enter:

   ```
   BuildProgram SQLSRV$DYNAMIC
   ```

   Alternatively, you can select Build from the Build menu and enter SQLSRV$DYNAMIC for the program name in the dialog box.
4. Press Enter to run the SQLSRV$DYNAMIC sample application.

3.2.5.2 Building the Sample Application in the THINK C Programming Environment

You can build the sample application on your Macintosh system by performing the following tasks:

1. Launch the THINK C Project Manager.
2. Create a project to build the Oracle SQL/Services sample application.

   If you have not previously created a project for building the sample, you must create a new TPM ANSI project.

   If a project for the sample already exists, recompile all the sources and then go to step 5.

3. Remove main.c by using the remove option from the Source menu.
4. Use the ADD option from the Source menu to add the following files to the project:
   - The sqlsrv$dynamic.c and sqlsrv$driver.c source files
   - The sqlsrv1, sqlsrv2, CommToolbox, MacTraps, and MacTraps2 libraries from the Sym C++ for Mac Folder:Macintosh Libraries:68K Libraries: folder

   Note

   You must create separate segments for the ANSI, sqlsrv1, and sqlsrv2 libraries (one segment for each library), due to THINK C program size limitations. Otherwise, you will receive this error message: "Code segment too big." Refer to the THINK C documentation for information about creating program segments.

5. Select the Build the Application entry from the Project menu to create the sample project. Call it SQLSRV$DYNAMIC.

3.2.6 Building the Sample Application on the Solaris Operating System

The source code for sqsdynu.c and sqsdrvu.c is available on line. To copy installation files to your default directory and to compile, link, and run the sample application, enter the following commands:

```
% cp /usr/sqlsrv/* .
% make -f sqsdynu.mak DNET=
% sqsdynu
```

Replace /usr/sqlsrv with the name of the directory in which you installed the Oracle SQL/Services Solaris client API kit. When linking the sample application, references to the DECnet library must be removed, as the DNET= argument indicates.

3.3 Running the Sample Application

When the sample executable program starts up, it prompts you for the information required to create an association with a remote system. When the association is made, the program prompts for SQL statements to execute. For example, on the OpenVMS operating system, this is what you would see:

```
$ run sqlsrv$dynamic
Server node OR SQL*Net service name: MYNODE
Network Transport: DECNET
Server account name: MYNAME
Server account password: ****
Service name [GENERIC]:

Enter any dynamically executable SQL statement,
continuing it on successive lines.
Terminate the statement with a semicolon.
Built-in commands are: [no]echo and exit.

SQL> ATTACH ’FILENAME sql_personnel’;
SQL> SELECT * FROM EMPLOYEES WHERE FIRST_NAME STARTING WITH ?;
Enter value for:   FIRST_NAME
Maximum length is: 10
DATA> Norman

------ BEGIN RESULT TABLE ------
EMPLOYEE_ID         : 00168
LAST_NAME           : Nash
FIRST_NAME          : Norman
MIDDLE_INITIAL      :
ADDRESS_DATA_1      : 87 West Rd.
ADDRESS_DATA_2      :
CITY                : Meadows
STATE               : NH
POSTAL_CODE         : 03587
SEX                 : M
BIRTHDAY            : 1932102300000000
STATUS_CODE         : 1
---------- END OF ROW ----------
   .
   .
   .
---------- END OF ROW ----------
EMPLOYEE_ID         : 00245
LAST_NAME           : Roberts
FIRST_NAME          : Norman
MIDDLE_INITIAL      : U
ADDRESS_DATA_1      : 162 Tenby Dr.
ADDRESS_DATA_2      :
CITY                : Chocorua
STATE               : NH
POSTAL_CODE         : 03817
SEX                 : M
BIRTHDAY            : 1949061100000000
STATUS_CODE         : 1
---------- END OF ROW ----------
------- END RESULT TABLE -------
SQL> exit;
$
```

To select the network transport, type D or DECnet to select the DECnet transport; type T or TCP to select the TCP/IP transport; type A or AppleTalk to select the AppleTalk-DECnet Gateway transport; type S or SQLNET to select the SQL*Net transport; or type N or NETWARE to select the NetWare IPX/SPX transport. Note that not all these transports are supported on all the client platforms and that all the transports supported by Oracle SQL/Services may not be installed on your node. See Table 1–1 for a list of the network transports supported for each client platform.

On Macintosh systems, you can use the transport selected in the Oracle SQL/Services control panel by typing the Enter key in response to the transport prompt. The sample does not prompt for a transport on the Solaris operating system because TCP/IP is the only transport supported by Oracle SQL/Services on that platform.

3.4 Driver Module

When a user runs the sample application, the flow of control is as follows:

- Call a routine to create an association. Although the driver creates only one association, Oracle SQL/Services allows an application to have several associations active at any given time.
- Enter a loop that inputs SQL statements and passes them to the execute_statement function for processing.
- Call a routine to close the association.

The implementation of the terminal input/output in the driver is unimportant. The module is intended to be easily replaced.

3.5 Dynamic Module

This section describes how the sample application works and provides some examples that illustrate how to call some of the more commonly used Oracle SQL/Services API routines.

3.5.1 Creating an Association

The sample program contains a function named create_association that does the following:

- Declares the variables required for creating an association.
  - Association identifier

  Most Oracle SQL/Services API routines require an association identifier that specifies for which association a call is being made. An association identifier is returned as an output argument during the successful completion of a call to the sqlsrv_associate API routine. The association identifier is then specified as an input argument to most of the other Oracle SQL/Services API routines, with the exception of some of the sqlsrv_sqlda_xxx and sqlsrv_sqlda2_xxx functional interface routines.

  In the sample application, the main routine in the driver module passes in the address of the association identifier, which it declares as follows:

  ```
  ASSOCIATE_ID    assoc_id;
  ```

  - Error message buffer

    If you do not specify an alternate error message buffer, Oracle SQL/Services uses the 70-byte SQLERRMC field in the SQLCA data structure. However, because the SQLERRMC field may not be long enough to hold all the possible error messages that can be returned by the Oracle SQL/Services server and Oracle Rdb, Oracle Corporation recommends that you allocate a larger message buffer for each association.

    In the sample application, the main routine in the driver module passes in the address of a 512-byte message buffer, which is sufficient for all possible messages. The driver routine declares the error buffer as follows:

    ```
    unsigned char    error_buf[512];
    ```

- Gets the node name, network transport, user name, password, and service name for the server system from the argument vector; if any of these are missing, the create_association function prompts the user.
- Sets up the association structure as follows:

  ```
  associate_str.VERSION = SQLSRV_V700;        /* Structure version number */
  associate_str.CLIENT_LOG = 0;               /* Disable client logging.  */
  associate_str.SERVER_LOG = 0;               /* Obsolete                 */
  associate_str.LOCAL_FLAG = 0;               /* Obsolete                 */
  associate_str.MEMORY_ROUTINE = NULL;        /* Use default memory rtns. */
  associate_str.FREE_MEMORY_ROUTINE = NULL;   /* Use default memory rtns. */
  associate_str.ERRBUFLEN = error_buf_len;    /* Alternate err buf length */
  associate_str.ERRBUF = error_buf;           /* Alternate error buffer   */
  associate_str.class_name = (CHARPTR)service_name; /* Service name       */
  associate_str.xpttyp = xpt;                 /* Transport type           */
  associate_str.attach = NULL;                /* No SQL ATTACH statement  */
  associate_str.declare = NULL;               /* No SQL DECLARE statement */
  associate_str.appnam = (CHARPTR)"Sample App"; /* Our application name   */
  ```

  This structure is described in detail in Section 7.2.

- Calls the API routine sqlsrv_associate to create the association.

```
sts = sqlsrv_associate(
            node_name,              /* node name.                      */
            user_name,              /* user name.                      */
            password,               /* password.                       */
            NULL,                   /* protocol read buffer.           */
            NULL,                   /* protocol write buffer.          */
            0,                      /* read buffer size.               */
            0,                      /* write buffer size.              */
            NULL,                   /* Let SQL/Services allocate SQLCA. */
            &associate_str,         /* ASSOCIATE structure.            */
            assoc_id                /* Association handle.             */
            );
```

By specifying the read and write buffer pointers as NULL and the read and write buffer lengths as zero, the sample application directs Oracle SQL/Services to allocate read and write buffers of the default size. By specifying a NULL SQLCA pointer, the sample application directs Oracle SQL/Services to allocate memory for the SQLCA structure. Note that by specifying the associate structure as Version 7.0, the sample application directs Oracle SQL/Services to process extensions to the original structure, which include the service (class) name, transport type, and application name fields.

Creating an association is a multiphase process, which starts with the Oracle SQL/Services client API validating the routine arguments, allocating memory for the association, establishing a network connection to the server, and so forth. Because a new association can fail for different reasons, client applications must be written to handle different types of failure.

If the Oracle SQL/Services client API detects any invalid arguments, it does not allocate any memory for the association, stores a NULL value in the association ID variable, and returns a single error status as the function return value. In this situation, the client application need perform no additional work to clean up the association; however, no additional error information is available.

If the routine arguments are valid, Oracle SQL/Services allocates memory for the association and attempts to connect to the server. Once the routine arguments have been successfully validated, Oracle SQL/Services always returns a non-NULL value in the association ID, even if the connection to the server is not established successfully. For example, perhaps a user typed an invalid password. In this situation, the client application can obtain additional error information by calling the sqlsrv_sqlca_error_text and sqlsrv_sqlca_error API routines. After retrieving any additional error information, the client application must then clean up the association by calling the sqlsrv_release API routine.

The sample application uses the following logic to handle the situation where a call to the sqlsrv_associate API routine fails:

```
if (sts != SQL_SUCCESS)
    {
    if (*assoc_id != NULL)
        {
        report_error(*assoc_id);
        sqlsrv_release(
                *assoc_id,          /* association ID.                  */
                NULL                /* reserved argument.               */
                );
        }
    else
        {
        report_sqlsvcs_error((SQS_LONGWORD)sts, 0, 0);
        }
    }
```

The report_error and report_sqlsvcs_error functions in the sample application are described in Section 3.5.2.10.

3.5.2 Processing the Dynamic SQL Statement

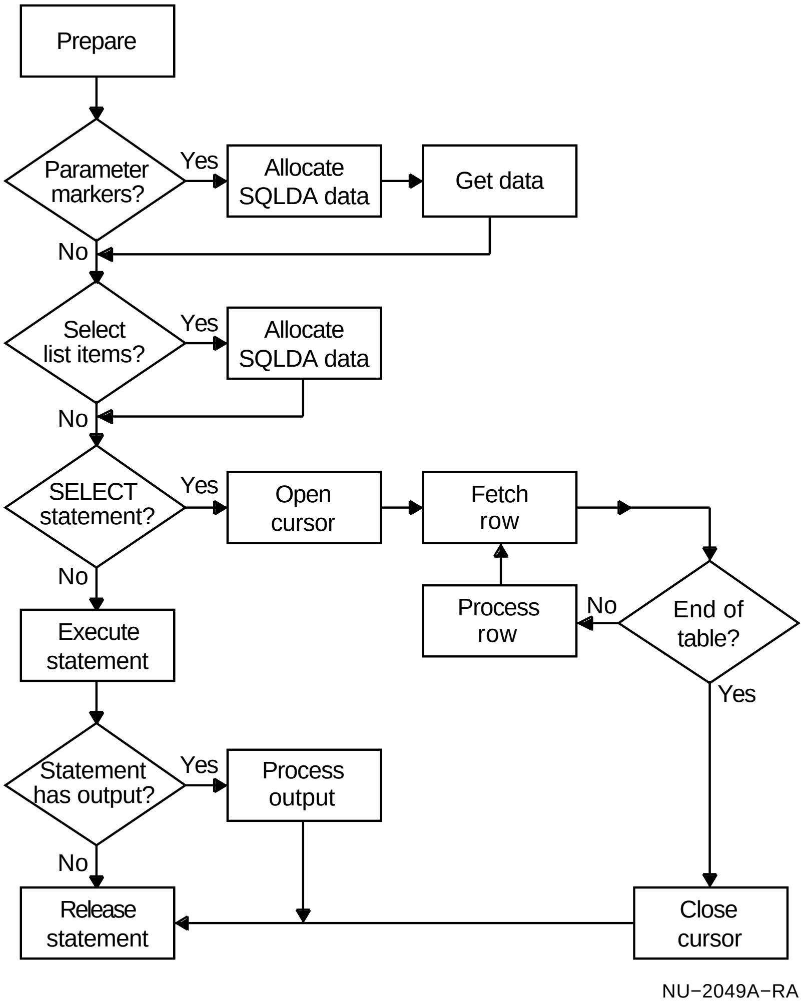
The sample program contains a function named execute_statement that processes the statement string passed to it by the driver module. As shown in Figure 3–1, the execute_statement function does the following:

- Checks for statements, such as COMMIT and ROLLBACK, that can be executed using the sqlsrv_execute_immediate API routine, processes them accordingly, and returns.
- Calls the sqlsrv_prepare API routine, which prepares the SQL statement and returns a statement ID.
- Calls the sqlsrv_sqlca_sqlerrd API routine to retrieve the SQLERRD array to obtain the statement type from the SQLERRD[1] array element.
- If the statement contains parameter markers, calls the sqlsrv_sqlda_allocate_data API routine to allocate memory for the data and indicator variables, then calls the get_params function to prompt for parameter marker values.
- If the statement contains select list items, calls the sqlsrv_allocate_sqlda_data API routine to allocate memory for the data and indicator variables.

- If the statement is a SELECT statement:
  - Calls the sqlsrv_open_cursor API routine to open a cursor
  - For each row in the result table, calls the sqlsrv_fetch API routine to fetch the row and calls the display_select_list routine to display the data
  - Calls the sqlsrv_close_cursor API routine to close the cursor
- If the statement is not a SELECT statement, calls the sqlsrv_execute_in_out API routine to execute the statement. If the statement has output, such as a singleton-SELECT statement or a CALL statement to a procedure with output or input/output arguments, calls the display_select_list function to display the data.
- Calls the sqlsrv_release_statement API routine to release the prepared statement.

Section 3.5.2.1 through Section 3.5.2.10 explain the workings of the execute_statement and get_params functions in more detail.

Figure 3–1 Statement Execution Flow



3.5.2.1 Declaring and Allocating SQLDA_ID Identifiers

The SQLDA structure contains SQL parameter marker and select list metadata as well as pointers to the data and indicator variables. The SQLDA_ID identifiers are the means by which your application and the Oracle SQL/Services API communicate about the SQL statement being prepared for execution.

Oracle SQL/Services applications must allocate variables that point to the SQLDA_ID identifiers. The execute_statement function contains the following declarations:

```
SQLDA_ID     param_sqlda;
SQLDA_ID     select_sqlda;
```

3.5.2.2 Executing SQL Statements Using the sqlsrv_execute_immediate API Routine

Simple SQL statements that do not contain parameter markers or select list items can be executed using the sqlsrv_execute_immediate API routine. The sample application checks for statements such as COMMIT and ROLLBACK, and executes them using the sqlsrv_execute_immediate API routine as follows:

```
sts = sqlsrv_execute_immediate(
            assoc_id,           /* association ID.                  */
            0,                  /* database id, must be zero.       */
            sql_statement       /* SQL statement.                   */
            );
```

Note

The sample application uses the sqlsrv_execute_immediate API routine to process SQL statements such as COMMIT and ROLLBACK in order to demonstrate how to use the sqlsrv_execute_immediate API routine. However, in a real application, where such statements may be used frequently, you should consider preparing such statements once and executing the prepared statements as needed.

3.5.2.3 Preparing the SQL Statement

All applications call the sqlsrv_prepare API routine to prepare an SQL statement. The sample application lets Oracle SQL/Services allocate memory for the parameter marker and select list SQLDA structures; therefore, it initializes the select_sqlda and param_sqlda SQLDA_IDs to NULL. For example:

```
select_sqlda = NULL;
param_sqlda = NULL;

sts = sqlsrv_prepare(
            assoc_id,           /* association ID.                   */
            0,                  /* database id, must be zero.        */
            sql_statement,      /* SQL statement.                    */
            &statement_id,      /* to receive prepared statement id  */
            &param_sqlda,       /* to receive parameter marker SQLDA */
            &select_sqlda       /* to receive select list SQLDA      */
            );
```

If the server successfully prepares the statement, it returns a statement ID to the client, which the Oracle SQL/Services client API stores in the statement_id variable. If an SQL statement contains either parameter markers or select list items, then the Oracle SQL/Services client API allocates memory for one or both SQLDAs and returns the memory pointer or handle to the application in the param_sqlda or select_sqlda variables.

The sample application calls the sqlsrv_sqlca_sqlerrd API routine to obtain the statement type from the SQLERRD[1] array element as follows:

```
sts = sqlsrv_sqlca_sqlerrd(
            assoc_id,          /* association ID.              */
            sqlerrd_array      /* to receive SQLERRD array     */
            );
   .
   .
   .
statement_type = sqlerrd_array[1];
```

3.5.2.4 Allocating Data and Indicator Variables

The sample application checks the param_sqlda and select_sqlda variables for non-NULL values to determine if the SQL statement contains any parameter markers or select list items. If any are present, the sample calls the sqlsrv_allocate_sqlda_data API routine to allocate memory for the data and indicator variables. For example, to allocate the data and indicator variables for a select list SQLDA:

```
if (select_sqlda != NULL)
    {
    sts = sqlsrv_allocate_sqlda_data(
                assoc_id,          /* association ID.               */
                select_sqlda       /* Select list SQLDA.            */
                );
   .
   .
   .
    }
```

If any parameter markers are present, the sample application also calls the get_params function, which is described in Section 3.5.2.5, to prompt the user for values for all parameter markers.

3.5.2.5 Processing Parameter Markers

The sample program includes a function named get_params that prompts the user for parameter markers. As in the driver module, the implementation of the terminal input/output is unimportant. As demonstrated in the get_params function, your application must perform the following steps:

1. Execute a loop that iterates once for each parameter marker in the SQL statement. The sqlsrv_sqlda_sqld API routine returns the number of parameter markers.

   ```
   for (i = 0; i < sqlsrv_sqlda_sqld( param_sqlda ); i++)
       {
      .
      .
      .
       }
   ```

2. Within the loop, call the sqlsrv_sqlda_ref_data API routine to obtain the data type and length, and pointers to the data and indicator variables for each parameter marker.

   ```
   sts = sqlsrv_sqlda_ref_data(
               param_sqlda,        /* parameter marker SQLDA       */
               i,                  /* column index number          */
               &coltyp,            /* to receive column data type  */
               &collen,            /* to receive column length     */
               &colscl,            /* to receive column scale/type */
               &coldata,           /* to receive column data ptr.  */
               &nullp,             /* to receive column ind. ptr.  */
               NULL                /* reserved argument            */
               );
   ```

3. Obtain a value for each parameter marker. The sample application checks that the user enters a data value that is not too long. To do so, it must check for certain data types and adjust the length returned by the sqlsrv_sqlda_ref_data API routine accordingly. For example, the length for the SQLSRV_GENERALIZED_DATE data type includes space for the null-terminator, so the maximum length must be decreased by 1, whereas the length for the SQLSRV_GENERALIZED_NUMBER data type does not include the additional 5 bytes that the sqlsrv_allocate_sqlda_data API routine allocates for integer values expressed in scientific notation, so the maximum length must be increased by 5. See Chapter 8 for more information on the data types supported by the Oracle SQL/Services client API.

```
switch (coltyp)
    {
    case SQLSRV_GENERALIZED_DATE:
        maxlen--;
        break;

    case SQLSRV_GENERALIZED_NUMBER:
        maxlen += 5;
        break;
    }
```

4. Set the indicator variable and store the value in the parameter marker's data variable according to each parameter marker's data type.

   To specify a NULL value for a parameter marker, store –1 in the indicator variable; otherwise, store 0 in the indicator variable.

   There are three fundamental data types in Oracle SQL/Services: fixed-length strings, null-terminated strings, and variable-length data with leading length field. Each Oracle SQL/Services data type maps to one of these fundamental data types. The sample application supports a subset of the full range of Oracle SQL/Services data types as follows.

   - Fixed-length strings

     There are two fixed-length data types: SQLSRV_LIST_VARBYTE (not supported by the sample application) and SQLSRV_ASCII_STRING. To store a fixed-length string in a parameter marker, the sample uses the memcpy C library function to copy the value, and the memset C library function to pad the value with spaces, if necessary.

     If your application calls the sqlsrv_allocate_sqlda_data API routine to allocate parameter marker variables, then Oracle SQL/Services allocates an extra byte of memory for parameter marker variables of type SQLSRV_ASCII_STRING. Therefore, your application can also use the strcpy C library function to copy a value to a parameter marker variable, because there is sufficient space for the trailing null-terminator. However, you should be aware that when Oracle SQL/Services sends fixed-length string parameter marker values to the server, it always sends the number of bytes specified by the parameter marker length in the SQLDA, regardless of the possible presence of a null-terminator anywhere in the string. Because Oracle SQL/Services does not treat fixed-length strings as a null-terminated string, the sample application always pads these values with spaces.

Note

The SQLSRV_LIST_VARBYTE, which is not supported by the sample application, is another instance of a fixed-length data type. However, because values of this data type can contain binary values, including null bytes, you should always use the memcpy C library function when processing values of this data type.

- Null-terminated strings

  There are three null-terminated data types: SQLSRV_GENERALIZED_NUMBER, SQLSRV_GENERALIZED_DATE, and SQLSRV_INTERVAL. To store a null-terminated string in a parameter marker, the sample simply uses the strcpy C library function to copy the value and null-terminator.

  If your application calls the sqlsrv_allocate_sqlda_data API routine to allocate parameter marker variables, Oracle SQL/Services always allocates the extra byte of memory required for the null-terminator. When Oracle SQL/Services sends null-terminated string parameter marker values to the server, it uses the strlen C library function to determine how much data to send.

- Variable-length data with leading length field

  There are two variable-length data types: SQLSRV_VARBYTE (not supported by the sample application) and SQLSRV_VARCHAR. To store a variable-length data value in a parameter marker, the sample first stores the length in the leading unsigned 16-bit length field, then uses the memcpy C library function to copy the data value.

  If your application calls the sqlsrv_allocate_sqlda_data API routine to allocate parameter marker variables, then Oracle SQL/Services allocates sufficient memory to accomodate the leading length field and a data value of the maximum length; however, it does not allocate space for a null-terminator. Therefore, your application should not use the strcpy C library function to copy a variable-length data value. When Oracle SQL/Services sends a variable-length data value to the server, it uses the leading length field to determine how much data to send.

Note

If your application uses the SQLDA2 format, the leading length field is an unsigned 32-bit integer.

See Chapter 8 for more information on the data types supported by the Oracle SQL/Services client API.

The following code example illustrates how the sample application processes each data type.

```
switch(coltyp)
    {

    case SQLSRV_ASCII_STRING:

        /* fixed-length string: copy the data to the        */
        /* SQLDATA memory; pad with spaces if necessary     */

        memcpy( (SCHARPTR)coldata, lcldata, len );
        if (len < maxlen)
            {
            memset( (SCHARPTR)coldata+len, ' ', maxlen-len );
            }
        break;

    case SQLSRV_GENERALIZED_NUMBER:
    case SQLSRV_GENERALIZED_DATE:
    case SQLSRV_INTERVAL:

        /* null-terminated strings: just use strcpy to      */
        /* copy the data value and the null-terminator      */

        strcpy( (SCHARPTR)coldata, lcldata );
        break;

    case SQLSRV_VARCHAR:

        /* variable-length data with length field: write    */
        /* the length into the leading 16-bit length field  */
        /* of the buffer, then advance the pointer over     */
        /* the length to the beginning of the data and      */
        /* copy the data                                    */

        varchar_ptr = coldata;
        *(unsigned short int *)varchar_ptr = len;
        varchar_ptr += sizeof(unsigned short int);
        memcpy( (SCHARPTR)varchar_ptr, lcldata, len );
        break;

    } /* switch */
```

5. After processing each parameter marker, call the sqlsrv_sqlda_unref_data API routine to de-reference the parameter marker's data and indicator variables.

```
sts = sqlsrv_sqlda_unref_data(
            param_sqlda,        /* parameter marker SQLDA       */
            i                   /* column index number          */
            );
```

3.5.2.6 Testing for SELECT Statements

To test for a SELECT statement, the sample application checks the statement_type variable, which it set previously using the SQLERRD[1] field of the SQLCA. Whenever Oracle Rdb SQL prepares an SQL statement, it stores the statement type in the SQLERRD[1] field, as follows:

- 0 = executable statement, excluding CALL statements
- 1 = SELECT statement
- 2 = CALL statement

3.5.2.7 Processing a SELECT Statement

To process a SELECT statement, the sample application opens a cursor, fetches and displays each row in the result table, then closes the cursor, as follows:

- Calls the sqlsrv_open_cursor API routine to open the cursor. Note that the sample has previously prompted the user for the values of any parameter marker values.

```
sts = sqlsrv_open_cursor(
            assoc_id,           /* association id                   */
            cursor_name,        /* cursor name                      */
            statement_id,       /* statement ID                     */
            param_sqlda         /* parameter marker SQLDA           */
            );
```

- For each row in the result table, calls the sqlsrv_fetch API routine to fetch the row, then calls the display_select_list function to display the values. See Section 3.5.2.9 for more information about the display_select_list function.

```
printf("------ BEGIN RESULT TABLE ------\n");
do
    {
    sts = sqlsrv_fetch(
                assoc_id,       /* association id                   */
                cursor_name,    /* cursor name                      */
                0,              /* scroll option                    */
                0L,             /* position                         */
                select_sqlda    /* select list SQLDA                */
                );
    switch (sts)
        {
        case SQL_SUCCESS:
            sts = display_select_list(assoc_id, select_sqlda);
            printf("---------- END OF ROW ----------\n");
            break;
            case SQL_EOS:
                printf("------- END RESULT TABLE -------\n");
                break;

            default:
                handle_error(assoc_id);
                break;
            }
        } while (sts == SQL_SUCCESS);
```

- Calls sqlsrv_close_cursor API routine to close the cursor when all rows have been fetched.

```
sts = sqlsrv_close_cursor(
            assoc_id,          /* association id                  */
            cursor_name        /* cursor name                     */
            );
```

3.5.2.8 Processing Executable Statements

To process an executable SQL statement, including CALL statements, the sample application calls the sqlsrv_execute_in_out API routine as follows:

```
sts = sqlsrv_execute_in_out(
            assoc_id,          /* association ID.                 */
            0,                 /* database_id, must be zero.      */
            statement_id,      /* Prepared statement id.          */
            SQLSRV_EXE_W_DATA,  /* Normal nonbatched execute mode.  */
            param_sqlda,       /* Parameter marker SQLDA.         */
            select_sqlda       /* Select list SQLDA.              */
            );
```

The sqlsrv_execute_in_out API routine handles any executable statement, regardless of whether the statement has any input in the form of parameter markers, or output in the form of select list items. See Section 4.1 for information on batched execution of executable statements with parameter markers.

If the executable SQL statement has any output, the sample application calls the display_select_list to display the values of the select list items.

3.5.2.9 Processing Select List Items

The sample application includes a function named display_select_list that displays the values of any select list items in a select list SQLDA. As in the driver module, the implementation of the terminal input/output is unimportant. As demonstrated in the display_select_list function, your application must perform the following steps.

1. Execute a loop that iterates once for each select list item in the SQL statement. The sqlsrv_sqlda_sqld API routine returns the number of select list items.

```
for (i = 0; i < sqlsrv_sqlda_sqld( select_sqlda ); i++)
    {
    .
    .
    .
    }
```

2. Within the loop, call the sqlsrv_sqlda_ref_data API routine to obtain the data type and length, and pointers to the data and indicator variables for each select list item.

   ```
   sts = sqlsrv_sqlda_ref_data(
               select_sqlda,       /* select list SQLDA            */
               i,                  /* column index number          */
               &coltyp,            /* to receive column data type  */
               &collen,            /* to receive column length     */
               &colscl,            /* to receive column scale/type */
               &coldata,           /* to receive column data ptr.  */
               &nullp,             /* to receive column ind. ptr.  */
               NULL                /* reserved argument            */
               );
   ```

3. Check the indicator variable and process the value in each select list item's data variable according to the data type.

   If the indicator variable for a select list item is set to –1, indicating a NULL value, the sample application displays "NULL" and proceeds to the next select list item. Otherwise, the sample application displays the data value based on the data type of the select list item.

   There are three fundamental data types in Oracle SQL/Services: fixed-length strings, null-terminated strings, and variable-length data with leading length field. Each Oracle SQL/Services data type maps to one of these fundamental data types. The sample application supports a subset of the full range of Oracle SQL/Services data types as follows.

   - Fixed-length strings

     There are two fixed-length data types: SQLSRV_LIST_VARBYTE (not supported by the sample application) and SQLSRV_ASCII_STRING. To process a fixed-length string in a select list item's data variable, use the length and pointer variables set by the sqlsrv_sqlda_ref_data API routine. The sample application passes both the length and the pointer as arguments to the printf C library function using the format string "%-.*s\n". Alternatively, to copy the same value to a local variable, the sample could call the memcpy C library function, again specifying the length and pointer variables as arguments.

If your application calls the sqlsrv_allocate_sqlda_data API routine to allocate select list item variables, then Oracle SQL/Services allocates an extra byte of memory for select list item data variables of type SQLSRV_ASCII_STRING. This allows Oracle SQL/Services to null-terminate a string value when it receives fixed-length string select list item data values from the server. Therefore, you can also treat variables of type SQLSRV_ASCII_STRING as null-terminated strings using, for example, the strcpy C library function.

Note

The SQLSRV_LIST_VARBYTE, which is not supported by the sample application, is another instance of a fixed-length data type. However, because values of this data type can contain binary values, including null bytes, you should always use the memcpy C library function when processing values of this data type.

- Null-terminated strings

  There are three null-terminated data types: SQLSRV_GENERALIZED_NUMBER, SQLSRV_GENERALIZED_DATE, and SQLSRV_INTERVAL. To display a null-terminated string from a select list item's data variable, the sample simply passes the data pointer as an argument to the printf C library function using the format string "%s\n". Alternatively, to copy the same value to a local variable, the sample could simply call the strcpy C library function, again specifying the pointer variable as an argument.

- Variable-length data with leading length field

  There are two variable-length data types: SQLSRV_VARBYTE (not supported by the sample application) and SQLSRV_VARCHAR. To process a variable-length data value in a select list item's data variable, the sample first uses a pointer to retrieve the length from the leading unsigned 16-bit length field, then advances the pointer past the length field to the data area of the variable. The sample application passes both the length and the data pointer as arguments to the printf C library function using the format string "%-.*s\n". Alternatively, to copy the same value to a local variable, the sample could call the memcpy C library function, again specifying the length and data pointer variables as arguments.

If your application calls the sqlsrv_allocate_sqlda_data API routine to allocate parameter marker variables, then Oracle SQL/Services allocates sufficient memory to accommodate the leading length field and a data value of the maximum length; however, it does not allocate space for a null-terminator. Therefore, your application should not use the strcpy C library function to copy a variable-length data value.

Note

If your application uses the SQLDA2 format, the leading length field is an unsigned 32-bit integer.

See Chapter 8 for more information on the data types supported by the Oracle SQL/Services client API.

The following code example illustrates how the sample application processes each data type.

```
/* check the indicator variable for NULL value */

if (*nullp < 0)
    {
    printf("NULL\n");
    }
else
    {
    switch (coltyp)
        {
        case SQLSRV_ASCII_STRING:

            /* Fixed-length character strings */

            printf("%-.*s\n", collen, coldata);
            break;

        case SQLSRV_GENERALIZED_NUMBER:
        case SQLSRV_GENERALIZED_DATE:
        case SQLSRV_INTERVAL:

            /* Null-terminated strings */

            printf("%s\n", coldata);
            break;

        case SQLSRV_VARCHAR:

            /* Counted string. The first 16-bit unsigned word of    */
            /* the data buffer is the length. Get length then       */
            /* advance the pointer to the data.                     */
```

```
                /* Note: SQLSRV_VARCHAR data may contain nonprintable   */
                /* binary data; a real application may not display the  */
                /* data value using printf.                             */

                varchar_ptr = coldata;
                varchar_len = *(unsigned short int *)varchar_ptr;
                varchar_ptr += sizeof(unsigned short int);
                printf("%-.*s\n", varchar_len, varchar_ptr);
                break;

            } /* switch */

        } /* else */
```

4. After processing each select list item, call the sqlsrv_sqlda_unref_data API routine to de-reference the select list item's data and indicator variables.

```
sts = sqlsrv_sqlda_unref_data(
            select_sqlda,       /* select list SQLDA            */
            i                   /* column index number          */
            );
```

3.5.2.10 Error Handling

The sample application contains three functions that handle error conditions.

- handle_error function

  The handle_error function is the main error handling routine for the sample application. It first calls the report_error function to display an error message. It then checks the error status and terminates the application if a nonrecoverable error occurred, such as a network error or if the server was shut down.

```
major_error = report_error(assoc_id);
if (major_error == SQLSRV_NETERR ||
    major_error == SQLSRV_INTERR ||
    major_error == SQLSRV_EXEINTERR ||
    major_error == SQLSRV_CONNTIMEOUT ||
    major_error == SQLSRV_SVC_SHUTDOWN)
    {
    sqlsrv_release(assoc_id, NULL);
#if defined (_WINDOWS)
    IvpExit();
#else
    exit(2);
#endif
    }
```

- report_error function

The report_error function is responsible for displaying an error message associated with the most recent error. It is called by the handle_error function and by the create_association function if an error occurred trying to connect to the server. The report_error function first calls the sqlsrv_sqlca_error_text API routine to retrieve any error text that might have been returned by the server or produced by the Oracle SQL/Services client API.

```
sts = sqlsrv_sqlca_error_text(
            assoc_id,           /* associate ID                        */
            &err_msg_len,       /* to receive error message length  */
            err_msg_buf,        /* error message buffer                */
            sizeof(err_msg_buf) /* size of error message buffer        */
            );
```

If an error message is available, the report_error function displays the error message text and returns. If no error message is available, the report_error function calls the sqlsrv_sqlca_error API routine to retrieve the major and minor error codes, then calls the report_sqlsvcs_error function to display an error message based on the error codes.

```
sts = sqlsrv_sqlca_error(
            assoc_id,           /* associate ID                        */
            &major_error,       /* to receive major error code         */
            &minor_error_1,     /* to receive first suberror           */
            &minor_error_2      /* to receive second suberror          */
            );
```

- report_sqlsvcs_error function

  The report_sqlsvcs_error function accepts as input major and minor error codes, then displays an error message based on those error codes. It is called by the report_error_function if no error message is available and called by the create_association if an error occurs trying to connect to the server and the sqlsrv_associate API routine does not return an association ID.

The execute_statement function checks the return status after calling every Oracle SQL/Services client API routine. If a call fails, the execute_statement function calls the handle_error function, calls the sqlsrv_release_statement API routine to release the prepared statement, then returns the failure status to the caller.

```
if (sts != SQL_SUCCESS)
    {
    handle_error(assoc_id);
    sqlsrv_release_statement(assoc_id, 1, &statement_id);
    return sts;
    }
```

Note that if a call to the sqlsrv_execute_immediate or sqlsrv_prepare API routines fails, there is no prepared statement to release.

3.5.2.11 Releasing Prepared Statements

When a prepared statement is no longer needed, the execute_statement function calls the API routine sqlsrv_release_statement to release the resources allocated for that statement:

```
sts = sqlsrv_release_statement(
                    assoc_id,           /* association handle.          */
                    1,                  /* no. of statement ids.        */
                    &statement_id       /* statement id array.          */
                    );
```

If your application prepares several statements at one time, you can release any or all of them together by passing an array of statement identifiers to the API routine sqlsrv_release_statement. The sample application prepares only one statement at a time; therefore, it passes the address of the statement ID variable to sqlsrv_release_statement. Effectively this is an array of one element.

4
Performance Considerations

This chapter describes how to improve the performance of your programming applications by reducing the number of client/server network messages required to perform operations.

4.1 Batched Execution

When your application executes a prepared INSERT, UPDATE, or DELETE statement that contains parameter markers, it can control whether the API sends one row or several rows of data at a time to the server for processing. Frequently, batched execution reduces the number of messages required to complete the operation.

The mechanism for controlling batched execution is the execute_flag parameter in the sqlsrv_execute_in_out routine, which is described in sqlsrv_execute_in_out. The values of the execute_flag parameter are shown in Table 6–7.

In batched execution, the API stores sets of parameter marker values in the message buffer until your application signals the end of the batched execution. If the message buffer becomes full during batched execution, the API sends the message to the server and begins a new message in a manner that is transparent to your application. In that case, when the batched parameter marker values arrive at the server, the server stores the values in a buffer until the application signals the end of the batched execution. If the application aborts the batched execution, the API clears the buffers on both the client and the server. Thus, the database remains consistent and there is no need to roll back the transaction.

In nonbatched execution, the API places each set of parameter marker values in the message buffer and sends the message to the server for execution.

Note

Once you initiate batched execution for a particular statement ID by calling the sqlsrv_execute_in_out API routine with the SQLSRV_EXE_BATCH flag, you cannot call other API routines or execute other statement IDs until you end batched execution for the current statement ID using the SQLSRV_EXE_WO_DATA, SQLSRV_EXE_W_DATA, or SQLSRV_EXE_ABORT flag.

The following example illustrates how to use the batched execution mechanism. Note that the error checking code has been removed from the example for brevity; however, your application should always check for and handle error conditions.

In this example, the application calls the prompt_for_order_details application function to prompt the user for new order details and to store the data into the parameter marker variables in the SQLDA.

As the user enters each line of the order, the application calls the sqlsrv_execute_in_out API routine with the SQLSRV_EXE_BATCH flag. This flag directs the Oracle SQL/Services client API to start or continue batched execution by queueing the row data for subsequent execution.

When the user has finished the order, the application calls the sqlsrv_execute_in_out API routine with the SQLSRV_EXE_WO_DATA flag to end batched execution. This flag directs the server to execute the previously queued requests, but does *not* send the data that is currently stored in the parameter marker SQLDA, which in this case would be the data from the most recent order line.

If the user cancels the order, the application calls the sqlsrv_execute_in_out API routine with the SQLSRV_EXE_ABORT flag to cancel batched execution without executing any previously queued requests.

```
    .
    .
    .
sql_statement = "INSERT INTO NEW_ORDERS VALUES ( ?, ?, ?, ?, ?, ?, ? )";
sts = sqlsrv_prepare(
            assoc_id,          /* association ID.                    */
            0,                 /* database id, must be zero.         */
            sql_statement,     /* SQL statement.                     */
            &statement_id,     /* to receive prepared statement id.  */
            &param_sqlda,      /* to receive parameter marker SQLDA. */
            &select_sqlda      /* to receive select list SQLDA.      */
            );

do
    {
    action = prompt_for_order_details( param_sqlda );
    switch ( action )
        {
        case ADD_ORDER_LINE:
            exec_flag = SQLSRV_EXE_BATCH;   /* Queue for later execution */
            break;

        case END_OF_ORDER:
            exec_flag = SQLSRV_EXE_WO_DATA; /* Execute queued requests   */
            break;

        case CANCEL_ORDER:
            exec_flag = SQLSRV_EXE_ABORT;   /* Cancel batched execution  */
            break;
        }

    sts = sqlsrv_execute_in_out(
                    assoc_id,           /* association ID.          */
                    0,                  /* reserved, must be zero.  */
                    statement_id,       /* Prepared statement id.   */
                    exec_flag,          /* Execute function flag.   */
                    param_sqlda,        /* Parameter marker SQLDA.  */
                    select_sqlda        /* Select list SQLDA.       */
                    );

    } while ( action == ADD_ORDER_LINE );

sts = sqlsrv_release_statement(
                assoc_id,           /* association ID.                  */
                1,                  /* number of statement id’s.        */
                &statement_id       /* statement id array.              */
                );
   .
   .
   .
```

Note

Alternatively, you can use the SQLSRV_EXE_W_DATA flag to end a batched execution operation. This flag directs the server to execute the previously queued requests, *including* the data that is currently stored in the parameter marker SQLDA.

4.2 Improving Row Fetch Performance

You can improve row fetch performance of your application by setting appropriate read and write buffer sizes for your client application based on the sizes of the data values. In addition, you can improve row fetch performance using the sqlsrv_fetch_many routine.

Setting Buffer Sizes

Oracle Corporation recommends that for a fetch-intensive application, in which you are using the sqlsrv_fetch_many routine and are working with large data values, such as images stored in lists (segmented strings), that you specify values greater than 1300 bytes for the read_buffer and write_buffer parameters in the sqlsrv_associate routine. You do this to ensure optimal performance for moving data between the server and client.

If you specify values greater than 5000 bytes for these two parameters in your application program, be sure to check that the server's dispatcher MAX_CLIENT_BUFFER_SIZE value is greater than these two parameter values. The default and minimum value allowed for the maximum client buffer size in a dispatcher process is 5000 bytes.

If the server's dispatcher MAX_CLIENT_BUFFER_SIZE is less than the read_buffer and write_buffer parameter values, the client picks the lower of the two sizes.

Fetching Multiple Rows

When your application fetches rows from a result table, it can control whether the server sends one row or several rows of data at a time to the API. Fetching multiple rows at a time generally reduces the number of client/server messages required to complete the operation.

Note

The Oracle SQL/Services NetWare client does not support fetching multiple rows. The sqlsrv_fetch_many routine, when used with the IPX/SPX transport, will always return a success status but will not initiate a multiple row fetch operation. Therefore, existing applications may call sqlsrv_fetch_many, but will not see the performance improvements normally associated with this call. The Oracle SQL/Services NetWare client does not batch result tuples to reduce the number of Oracle SQL/Services messages, due to SPX flow control limitations.

The mechanism for fetching multiple rows is the sqlsrv_fetch_many routine, which is described in sqlsrv_fetch_many. Using the sqlsrv_fetch_many API routine to initiate a fetch many operation is as follows. Call the routine after calling the sqlsrv_open_cursor routine before the first call to the sqlsrv_fetch routine. The repeat_count parameter specifies the number of rows that the server can send to the API the next time your application calls sqlsrv_fetch. When you specify a repeat count of 0, the server continously fetches rows from the result table and transmits them to the client until all rows have been fetched. When you specify a repeat_count other than 0, your application must call the sqlsrv_fetch_many routine again once the specified number of rows have been fetched. You can call the sqlsrv_close_cursor API routine at any time to end a multiple row fetch operation.

Oracle Corporation recommends that you set the repeat_count to 0 if all rows are to be fetched. When the sqlsrv_fetch_many routine is called with a repeat_count of 0, all rows in the result table can be accessed with successive calls to sqlsrv_fetch. In this situation, the sqlsrv_fetch_many routine does not need to be called again. Oracle SQL/Services manages the message buffer transparently by filling each message buffer with as many rows as possible whenever the data in the buffer is exhausted by a sqlsrv_fetch call. Each successive call to the sqlsrv_fetch API routine retrieves the next row of data from the message buffer. When all the rows have been read from the buffer, the client API posts a network receive to read the next buffer from the server without having to send a fetch request to the server. When the specified number of rows has been fetched or when the last row in the table has been fetched, the API returns to the default behavior.

The sqlsrv_fetch_many API routine is responsible for configuring the Oracle SQL/Services API to begin a multiple row fetch operation; however, it does not fetch any rows. The multiple row fetch operation is not actually started until the application calls the sqlsrv_fetch API routine. Therefore, the sqlsrv_fetch_many API routine returns a success status even if no rows are in the result table.

Note

Once you initiate a multiple row fetch operation by calling the sqlsrv_fetch_many API routine, you cannot call other API routines until the specified number of rows or all rows have been fetched. The only exception is the sqlsrv_close_cursor API routine, which you can call to end a multiple row fetch operation. For this reason, and because the position of the cursor within the result table at the server is always ahead of the number of rows fetched by the client when a multiple row fetch operation is active, you cannot call the sqlsrv_execute_in_out API routine to execute statements such as INSERT . . . WHERE CURRENT OF cursor_name, UPDATE . . . WHERE CURRENT OF cursor_name, or DELETE . . . WHERE CURRENT OF cursor_name when a multiple row fetch is active.

The following example extends the sample application described in Chapter 3 to use the sqlsrv_fetch_many API routine. In this example, note that the only change to the logic is the addition of the call to the sqlsrv_fetch_many API routine; the rest of the routine remains the same.

```
   .
   .
   .
sts = sqlsrv_open_cursor(
            assoc_id,           /* association id                  */
            cursor_name,        /* cursor name                     */
            statement_id,       /* statement ID                    */
            param_sqlda         /* parameter marker SQLDA          */
            );
   .
   .
   .
sts = sqlsrv_fetch_many(
            assoc_id,           /* association id                  */
            cursor_name,        /* cursor name                     */
            1,    /* Row increment      */
            0     /* Fetch all rows      */
            );
   .
   .
   .
printf("------ BEGIN RESULT TABLE ------\n");
do
    {
    sts = sqlsrv_fetch(
                assoc_id,       /* association id                  */
                cursor_name,    /* cursor name                     */
                0,              /* direction                       */
                0L,             /* row number                      */
                select_sqlda    /* select list SQLDA               */
                );
    switch (sts)
        {
        case SQL_SUCCESS:
            sts = display_select_list(assoc_id, select_sqlda);
            printf("---------- END OF ROW ----------\n");
            break;

        case SQL_EOS:
            printf("------- END RESULT TABLE -------\n");
            break;
        default:
            handle_error(assoc_id);
            break;
        }
    } while (sts == SQL_SUCCESS);
   .
   .
   .
sts = sqlsrv_close_cursor(
            assoc_id,           /* association id                   */
            cursor_name         /* cursor name                      */
            );
   .
   .
   .
```

4.3 Using Stored Procedures and Compound Statements

A stored procedure is a set of operations performed on an Oracle Rdb database by one or more SQL statements that execute as a unit to perform a wide variety of database operations. The stored procedure resides within a stored module that is the object of compilation and encapsulates an operation, such as updating, deleting, or adding information to a table. The stored module resides as a schema object inside an Oracle Rdb database and defines at least one stored procedure. Stored procedures allow you to place an operation (or set of operations) in the database for reference by other users.

With client/server processing, your client system applications can attain much better performance by calling a set of stored procedures that reside on the server system. The stored procedures perform an operation or a series of operations on the database from the server side rather than locally storing and maintaining database requests containing the same SQL statements from the client side. With stored procedures, multiple SQL statements can be processed with a single CALL statement. This is useful if certain transactions are executed frequently. In such a case, the stored procedure can be created in advance on the server and called as needed by the client. Therefore, use stored procedures whenever possible.

Beginning with Oracle Rdb V7.0, you can dynamically prepare and execute compound statements using the dynamic SQL interface. A compound statement is a set of one or more SQL statements delimited by BEGIN and END statements. The SQL statements contained in a compound statement can contain parameter markers, select list items, or both. For example:

```
BEGIN
SET TRANSACTION READ WRITE;
INSERT INTO EMPLOYEES VALUES ( ?, ?, ?, ?, ?, ?, ?, ?, ? );
INSERT INTO SALARY HISTORY VALUES ( ?, ?, ?, ?, ? );
SELECT AVG( SALARY ), SUM( SALARY ) INTO ?, ? FROM EMPLOYESS;
COMMIT;
END
```

Compound statements have some of the same performance advantages as stored procedures, because a series of SQL statements can be executed at the server with a single call to the sqlsrv_execute_in_out API routine. In some situations, compound statements have an advantage over stored procedures brcause they can be constructed dynamically by an application as and when required. However, it is more efficient to use a stored procedure if a particular set of SQL statements are executed frequently. Furthermore, an application must have precise knowledge of the order of all parameter markers and select list items because parameter marker names and select list item names are not returned by Oracle Rdb when you prepare a compound statement.

For more information on using stored procedures and compound statements, see the *Oracle Rdb7 Guide to SQL Programming*.

4.4 Reusing SQL Statements

A prepared SQL statement should not be released when the statement can be reused. After a statement is prepared, the statement can be executed many more times with the same statement_id (see the sqlsrv_prepare and sqlsrv_execute_in_out routines for more information). This not only reduces the number of network messages, but also reduces resource consumption by not performing extra sqlsrv_prepare and sqlsrv_release routine calls. The only disadvantage is that extra memory will be needed to keep these prepared statements before they are released. For example:

```
sts = sqlsrv_prepare(
          assoc_id,
          0L,
          sql_statement_1,
          &statement_id_1,
          &param_sqlda_1,
          &select_sqlda_1,
          );
sts = sqlsrv_prepare(
          assoc_id,
          0L,
          sql_statement_2,
          &statement_id_2,
          &param_sqlda_2,
          &select_sqlda_2,
          );
```

```
 .
 .
 .
do {
  GetUserChoice(&choice);
  switch (choice)
    case CHOICE_1:
       sts = sqlsrv_execute_in_out(
                 assoc_id,
                 0L,
                 statement_id_1,
                 execute_flag,
                 param_sqlda_1,
                 select_sqlda_1
                 );
       if (sts != SQL_SUCCESS) {
          /*
            error condition
          */
          }
       break;
    case CHOICE_2:
       sts = sqlsrv_execute_in_out(
                 assoc_id,
                 0L,
                 statement_id_2,
                 execute_flag,
                 param_sqlda_2,
                 select_sqlda_2
                 );
       if (sts != SQL_SUCCESS) {
          /*
            error condition
          */
          }
       break;
 .
 .
 .
    default:
 .
 .
 .
    } /* switch (choice) */
 .
 .
 .
} while (choice != END_OF_CHOICE);
sts = sqlsrv_release_statement(
          assoc_id,
          1,
          &statement_id_1
          };
sts = sqlsrv_release_statement(
          assoc_id,
          1,
          &statement_id_2
          };
 .
 .
 .
```

5

Logging for Performance and Debugging

This chapter describes how to use client logging to help debug and monitor the performance of Oracle SQL/Services applications. Logging can be useful in debugging an application to verify that the application is sending the correct data to the server. Logging can be useful in tuning the performance of an application to set the network buffer size so that frequently sent messages that fit into a single network packet and do not have to be split into multiple packets.

5.1 Enabling and Disabling Logging

You enable client logging by setting one or more logging flags in the CLIENT_LOG field in the association structure (see Section 7.2) before calling sqlsrv_associate or by using one of the following operating system-specific mechanisms:

- All Windows operating systems

  Set the ClientLogging option to the appropriate value in the sqsapiw.ini or sqsapi32.ini file before running the application. For example:

  ```
  ClientLogging=7
  ```

- OpenVMS operating system

  Define the SQLSRV$CLIENT_LOG logical name using the appropriate value before running the application. For example:

  ```
  $ DEFINE SQLSRV$CLIENT_LOG 7
  ```

- All UNIX operating systems

  Set the SQLSRV_CLIENT_LOG environment variable to the appropriate value before running the application. For example:

  ```
  % setenv SQLSRV_CLIENT_LOG 7
  ```

You must use the CLIENT_LOG field in the association structure to enable logging on the Macintosh operating system.

Table 5–1 shows all the logging flag names and their numeric values.

Table 5–1 Client Logging Flags and Values

Flag Name	Value	Description
SQLSRV_LOG_DISABLED	0	Disables logging (default).
SQLSRV_LOG_ASSOCIATION	1	Enables association logging.
SQLSRV_LOG_ROUTINE	2	Enables API routine logging.
SQLSRV_LOG_PROTOCOL	4	Enables message protocol logging.
SQLSRV_LOG_SCREEN[1]	8	Sends logging to standard output on the client system as well as to the log file.
SQLSRV_LOG_OPNCLS	64	Opens and closes the log file around each log file write and is useful if a client is terminating abnormally while calling an Oracle SQL/Services client API routine. If the client process is terminating due to an unhandled error condition in an Oracle SQL/Services client API service, then it may be necessary to use the SQLSRV_LOG_OPNCLS option in order to write as much information as possible to the log file during every call to an Oracle SQL/Services client API service.
SQLSRV_LOG_FLUSH	128	Flushes pending output to the log file only at the end of each complete association-level, routine-level, and protocol-level log entry and is useful if a client application is terminating abnormally while executing application code. If the client process is terminating due to an unhandled error condition in the client application, use the SQLSRV_LOG_FLUSH option to flush pending output to the client log before each call to an Oracle SQL/Services client API service completes.
SQLSRV_LOG_BINARY	256	Dumps memory in structured format if data contains nonprintable characters.

[1] The SQLSRV_LOG_SCREEN flag is ignored on all Macintosh and Windows platforms.

To enable more than one type of logging, add the appropriate constants. For example:

```
associate_str.CLIENT_LOG = SQLSRV_LOG_ROUTINE + SQLSRV_LOG_SCREEN;
```

Most of the operating systems supported by the Oracle SQL/Services client API do not support multiple versions of the same file. However, sometimes it is necessary or advantageous to preserve the client log files produced by multiple associations. For example, Microsoft Access frequently uses two associations to process user requests. Therefore, Oracle SQL/Services uses the following algorithm to construct a unique client log file name to retain multiple client log files:

1. Use client.log if there is no existing log file named client.log.
2. Using client<nn>.log as a template, increment nn from 00 to 99 looking for a log file name for which there is no existing log file. For example client00.log, client01.log, and so forth. Use the first available unused file name.
3. If client.log and client00.log through client99.log all exist, use client.log, overwriting the existing client.log file.

Using this algorithm, Oracle SQL/Services can retain up to 101 client log files. Client log files can consume large amounts of disk space, depending on the application. Therefore, you may want to delete or archive log files once you have finished monitoring or debugging an application.

5.2 Association Logging

Association logging occurs whenever a client/server association is created, terminated, or aborted. Use this type of logging to debug server access in application programs.

Depending on the API routine called, association log entries include some or all of the following items:

❶ A header that identifies the entry as ASSOCIATE LEVEL LOG

❷ The name of the API routine

❸ The association identifier

❹ The name of the server node

❺ The name of the user account on the server

❻ The error status for the API routine

❼ The detailed error code for network or server errors

❽ The type of network transport used for client/server communication: DECnet, TCP/IP, AppleTalk, IPX/SPX, or SQL*Net

For example:

```
ASSOCIATE LEVEL LOG ❶
----SQLSRV_ASSOCIATE ❷
--------SQLSRV_ASSOCIATE ID: 7ac50 ❸
--------NODE: abcdef, ❹ USERNAME: xxxxxx, ❺ SQLCODE: 0, ❻ SQLERRD[0] 0 ❼
--------NETWORK TRANSPORT: DECnet ❽
```

These messages indicate that an association with a server system was created normally.

5.3 Routine Logging

Routine logging occurs whenever your application calls an Oracle SQL/Services API routine. Use this type of logging to debug execution flow in application programs.

Routine log entries include some or all of the following items:

❶ A header that identifies the entry as ROUTINE LEVEL LOG and contains a timestamp

❷ The name of the API routine

❸ The length in bytes of the SQL statement string

❹ The SQL statement string

❺ The name of the cursor

❻ The SQL statement identifier

❼ The execution flag

For example:

```
ROUTINE LEVEL LOG at 07:57:08 on 15-MAY-1996 ❶
----SQLSRV_PREPARE ❷
--------SQL STATEMENT
------------len: 45, ❸ value: Select * from sqlsrv_table
                                 where USERNAME = ? ❹

ROUTINE LEVEL LOG at 07:57:08 on 15-MAY-1996
----SQLSRV_OPEN_CURSOR
--------CURSOR NAME
------------sqlsrv_cursor ❺
--------STATEMENT ID
             1199896 ❻

ROUTINE LEVEL LOG at 07:57:08 on 15-MAY-1996
----SQLSRV_EXECUTE_IN_OUT
--------STATEMENT ID
------------1199897
--------EXECUTE FLAG:SQLSRV_EXE_W_DATA ❼
      .
      .
      .
```

Routine log entries that follow the sqlsrv_prepare routine also include metadata:

❶ The type of SQLDA (parameter marker or select list)

❷ The number of parameter markers or select list items

❸ The Oracle SQL/Services data type

❹ For character data types, the length of the data variable

❺ For numeric and date-time data types, the length of the data variable and the scale factor or type of date or interval, respectively (see Section 7.6)

❻ The name of the column

For example:

```
ROUTINE LEVEL LOG at 07:57:08 on 15-MAY-1996
----SELECT LIST SQLDA ❶
--------SQLDA: SQLD 4 ❷
--------[0].SQLTYPE: SQLSRV_ASCII_STRING, ❸ SQLLEN: 33 ❹
------------SQLNAME: USERNAME
--------[1].SQLTYPE: SQLSRV_GENERALIZED_NUMBER, SIZE 11, SCALE 0 ❺
------------SQLNAME: INTEGER_VALUE ❻
--------[2].SQLTYPE: SQLSRV_GENERALIZED_NUMBER, SIZE 24, SCALE 0
------------SQLNAME: DOUBLE_VALUE
--------[3].SQLTYPE: SQLSRV_GENERALIZED_DATE, SIZE 17, TYPE 0
------------SQLNAME: DATE_VALUE
```

Routine log entries that follow the sqlsrv_fetch, sqlsrv_open_cursor, and sqlsrv_execute_in_out routines also include data:

❶ The type of SQLDA (parameter marker or select list)

❷ The number of parameter markers or select list items

❸ The Oracle SQL/Services data type

❹ The value of the indicator variable

❺ The length of the value of the data variable

❻ The value of the data variable

For example:

```
ROUTINE LEVEL LOG at 07:57:08 on 15-MAY-1996
----SELECT LIST SQLDA ❶
--------SQLDA: SQLD 4 ❷
------------[0].SQLTYPE: SQLSRV_ASCII_STRING, ❸ SQLIND: 0 ❹
----------------len: 32, ❺ value: xxxxxx ❻
------------[1].SQLTYPE: SQLSRV_GENERALIZED_NUMBER, SQLIND: 0
----------------len: 1, value: 1
------------[2].SQLTYPE: SQLSRV_GENERALIZED_NUMBER, SQLIND: 0
----------------len: 23, value:  1.280000000000000E+002
------------[3].SQLTYPE: SQLSRV_GENERALIZED_DATE, SQLIND: 0
----------------len: 16, value: 1988070100000000
```

5.4 Message Protocol Logging

Message protocol logging occurs whenever a message is transmitted between the client API and the server process. Use this type of logging to verify that the Oracle SQL/Services client/server communications protocol is working as expected and to determine if request or response messages are being split into multiple network packets.

Protocol log entries include some or all of the following items:

❶ A header that identifies the entry as PROTOCOL LEVEL and contains a timestamp

❷ The word CLIENT to indicate where the log file was written

❸ The word “read” or “write” to indicate whether the packet was received or transmitted, respectively

❹ The timestamp

❺ The packet identification number, which is incremented from 0 from the beginning of the association

❻ The packet sequence number, which is nonzero in the following instances:

- Batched execution
- Multiple row fetches
- Any message that is too large for a single packet

❼ The message tag, which indicates a function to be executed on the server, an acknowledgment (ACK) that a function was executed successfully, or an error (ERROR) message

❽ Tags that represent routine parameters, including:

❾ The total length in bytes of the data

❿ The number of bytes of data in this packet

⓫ The data value

⓬ Subtags that describe SQLDA structures; indicates whether an SQLDA(1) or SQLDA2 is being used

For example:

```
PROTOCOL LEVEL LOG ❶ CLIENT: ❷  write (logonly)❸ at 07:57:08 on 15-MAY-1996 ❹
----PACKET ID: 11, ❺  PACKET SEQUENCE: 0 ❻
--------SQLSRV_FETCH ❼
------------STATEMENT ID ❽
--------------------len: 4,  ❾ value: 1000001 ❿
--------END OF MESSAGE

PROTOCOL LEVEL LOG CLIENT: read at 07:57:08 on 15-MAY-1996
----PACKET ID: 11, PACKET SEQUENCE: 0
--------SQLSRV_FETCH ACK
------------FETCH ROW NUMBER
--------------------len: 4, value: 3
------------SELECT LIST DATA ❼
----------------len: 2, value: 4
------------SQLVAR INDEX SQLDATA SQLIND ⓫
----------------SQLSRV_SQLVAR_INDEX  ⓫
--------------------len: 2, value: 0
----------------SQLSRV_SQLVAR_SQLIND1 ⓫
--------------------len: 2, value: 0
----------------SQLSRV_SQLVAR_SQLDATA1, len: 32 ⓫
--------------------len: 32, value: SMITH
```

```
------------SQLVAR INDEX SQLDATA SQLIND
----------------SQLSRV_SQLVAR_INDEX
--------------------len: 2, value: 1
----------------SQLSRV_SQLVAR_SQLIND1
--------------------len: 2, value: 0
----------------SQLSRV_SQLVAR_SQLDATA1, len: 1
--------------------len: 1, value: 3
      .
      .
      .
--------END OF MESSAGE
```

To determine the data type of an SQLDATA value, review the SQLDA information from the routine level log that is written at prepare time. For example:

```
ROUTINE LEVEL LOG at 07:57:08 on 15-MAY-1996
----SELECT LIST SQLDA
--------SQLDA: SQLD 2
--------[0].SQLTYPE: SQLSRV_ASCII_STRING, SQLLEN: 15
------------SQLNAME: EMPLOYEE_NAME
--------[1].SQLTYPE: SQLSRV_GENERALIZED_NUMBER, SIZE 6, SCALE 0
------------SQLNAME: COST_CENTER
      .
      .
      .
```

The following information is logged in the ASSOCIATION ACK message for the protocol level log:

❶ A header that identifies the entry as PROTOCOL LEVEL LOG CLIENT and contains a timestamp

❷ The name of the API routine

❸ The version of SQL used by the server

❹ The version of Oracle Rdb used by the server

❺ The server protocol version number

❻ The version of the server

❼ The process ID (PID) of the executor

❽ A flag to indicate the service attributes

❾ The maximum server buffer size

For example:

```
   .
   .
   .
PROTOCOL LEVEL LOG CLIENT: read ❶ at 07:57:08 on 15-MAY-1996
----PACKET ID: 1, PACKET SEQUENCE: 0
--------SQLSRV_ASSOCIATE ACK ❷
------------DEC SQL VERSION ❸
----------------SQLSRV_ASCII_STRING, len: 7
--------------------len: 7, value: V7.0-01
------------RDB ENG VERSION ❹
----------------SQLSRV_ASCII_STRING, len: 7
--------------------len: 7, value: V7.0-01
------------SERVER PROTOCOL VERSION ❺
----------------len: 2, value: 14
------------SQLSRV SRV VERSION ❻
----------------SQLSRV_ASCII_STRING, len: 7
--------------------len: 7, value: V7.0-01
------------EXECUTOR PID ❼
--------------------len: 4, value: 727725642
------------SERVICE ATTRIBUTES ❽
----------------len: 2, value: 0
------------MAXIMUM SERVER BUFFER SIZE
----------------len: 4, value: 5000 ❾
--------END OF MESSAGE
   .
   .
   .
```

These messages indicate that an association is made between a client and a server with a protocol of 14, using V7.0 of SQL and V7.0 of Oracle Rdb, using an executor whose process ID is 727725642 (decimal), using a universal (nondatabase) service, and a maximum server buffer size of 5000 bytes. This information can be beneficial in resolving server-related environmental issues and protocol version issues.

6
API Routines

This chapter describes the routines in the Oracle SQL/Services client application programming interface (API).

6.1 Documentation Format

Each Oracle SQL/Services API routine is documented using a structured format called the routine template. Table 6–1 lists the sections of the routine template, along with the information that is presented in each section and the format used to present the information. Some sections require no further explanation beyond what is given in Table 6–1. Those that require additional explanation are discussed in the subsections that follow the table.

Table 6–1 Sections in the Routine Template

Section	Description
Routine Name	Appears at the top of the page, followed by the English name of the routine
Overview	Appears directly below the routine name and explains, usually in one or two sentences, what the routine does
C Format	Shows the C function prototype from the include file sqlsrv.h
Parameters	Provides detailed information about each parameter
Notes	Contains additional pieces of information related to applications programming
Errors	Lists the Oracle SQL/Services errors that can occur in the routine

6.1.1 Routine Name

The Oracle SQL/Services API routine names are shown in the form sqlsrv_xxx, sqlsrv_sqlca_xxx, sqlsrv_sqlda_xxx, or sqlsrv_sqlda2_xxx, throughout the manual.

6.1.2 Return Values

The Oracle SQL/Services routine template does not include a “Returns” section. Except where explicitly noted, the Oracle SQL/Services API routines return a signed longword integer containing one of the values shown in Table 6–2.

Table 6–2 API Return Values

Return Value	Description
n = SQL_SUCCESS[1]	The routine completed successfully.
n < SQL_SUCCESS	An error occurred during processing. Refer to the SQLCA.SQLCODE for the specific error.
n > SQL_SUCCESS	A warning was issued during processing. Refer to the SQLCA for additional information.

[1] The symbol SQL_SUCCESS is defined as 0 in the include file sqlsrv.h.

6.1.3 C Format Section

The C Format section shows the function prototypes for the Oracle SQL/Services API routines exactly as they are declared in the include file sqlsrv.h.

6.2 Oracle SQL/Services Data Types

Table 6–3 lists the data types used in Oracle SQL/Services API routine calls and structures.

Table 6–3 API Parameter Data Types

Data Type	Description
ASSOCIATE_ID	An identifier that uniquely distinguishes one association from all others
ASSOCIATE_STR	Structure that specifies association characteristics
character string	Pointer to a null-terminated character string of type char
CHARPTR	Pointer to a buffer or variable of type unsigned char
PTRCHARPTR	Pointer to a variable of type CHARPTR
SHORTPTR	Pointer to a variable of type short

(continued on next page)

Table 6–3 (Cont.) API Parameter Data Types

Data Type	Description
LONGPTR	Pointer to a variable of type SQS_LONGWORD
PTRSHORTPTR	Pointer to a variable of type short * or SHORTPTR
PTRLONGPTR	Pointer to a variable of type LONGPTR
word (signed)	16-bit signed integer
word (unsigned)	16-bit unsigned integer
longword (signed)	32-bit signed integer
longword (signed) array	Array of signed 32-bit integers
longword (unsigned)	32-bit unsigned integer
pointer	An address whose size is platform specific
SQLDA_ID	An identifier (pointer or handle) used to access SQLDA data and metadata information
void	Arguments described with the void data type are reserved for future use
SQLCA_ID	An identifier (pointer or handle) used to access the data and structure SQLCA

To facilitate the development of portable Oracle SQL/Services client software modules, the following two 32-bit integer data types are type defined in the sqlsrv.h file and may be used to define variables in your programs:

```
SQS_LONGWORD            32-bit signed longword
SQS_UNSIGNED_LONGWORD   32-bit unsigned longword
```

6.3 API Routines

This section describes each of the API routines.

6.3.1 Association Routines

Association routines create and terminate client/server associations and control the association environment. Association routines include the following routines:

- sqlsrv_abort routine (see sqlsrv_abort)
- sqlsrv_associate routine (see sqlsrv_associate)
- sqlsrv_get_associate_info routine (see sqlsrv_get_associate_info)
- sqlsrv_release routine (see sqlsrv_release)

sqlsrv_abort

The sqlsrv_abort routine drops the network link between the client and server, frees client association resources, and rolls back active transactions on the server.

C Format

```
extern int sqlsrv_abort(
               ASSOCIATE_ID associate_id);
```

Parameters

associate_id

An identifier used to distinguish one active association from all others.

Errors

SQLSRV_INTERR	Internal error.
SQLSRV_INVASC	Invalid association identifier.

sqlsrv_associate

The sqlsrv_associate routine creates a network link between your application and the server, using the node name, user name, and password input parameters. It creates an association identifier used in subsequent routine calls and optionally binds specific input parameters, such as the message protocol buffers and SQLCA structure, to the association.

C Format

```
extern int sqlsrv_associate(
                char *node_name,
                char *user_name,
                char *password,
                CHARPTR read_buffer,
                CHARPTR write_buffer,
                SQS_LONGWORD read_buffer_size,
                SQS_LONGWORD write_buffer_size,
                SQLCA_ID *sqlca_str,
                struct ASSOCIATE_STR *associate_str,
                ASSOCIATE_ID *associate_id);
```

Parameters

node_name

Address of a null-terminated string containing the name of the server node. If you are using the SQL*Net transport, this parameter specifies either the SQL*Net Service Name or the SQL*Net Alias.

user_name (optional)

Address of a null-terminated string containing the user name that the server uses to authenticate the user and determine if the user is authorized to access the specified service. If this parameter is NULL, the DECnet transport is selected, and the client is connecting to a server on an OpenVMS node, then the server looks for an Oracle SQL/Services proxy for the client node name. If there is no proxy for the client node, or a transport other than DECnet is selected, the server checks to see if unknown users are authorized to access the specified service. If unknown users are not authorized to access the service, the association fails. See the *Oracle SQL/Services Server Configuration Guide* for more information on client authentication and authorization, and how Oracle SQL/Services uses the client user name.

password (optional)

Address of a null-terminated string containing the corresponding password to the specified user name. You must include a password when you specify a user name.

read_buffer (optional)

Address of a buffer of type unsigned char used by the API to receive messages from the server. If you specify a buffer address of NULL, Oracle SQL/Services allocates the buffer. Oracle Corporation recommends that you pass a NULL value.

write_buffer (optional)

Address of a buffer of type unsigned char used by the API to build messages to send to the server. If you specify a buffer address of NULL, Oracle SQL/Services allocates the buffer. Oracle Corporation recommends that you pass a NULL value.

read_buffer_size (optional)

The size in bytes of the read_buffer. If a read_buffer is passed, the read_buffer_size must contain its size. The minimum value is 256 bytes. If no read_buffer is passed, Oracle SQL/Services allocates a buffer of size read_buffer_size if the parameter is non-zero, or of a default size if the parameter is zero. See Table 6–4 for valid combinations of buffer-related parameters. The values for read_buffer_size and write_buffer_size must be equal. This is true for both user-allocated buffers, or when the application requests that Oracle SQL/Services allocate buffers of a specified size.

Table 6–4 Valid Combinations of Buffer-Related Parameters for the sqlsrv_associate Routine

Transport/Platform	Buffer Specified	Buffer Size Specified	Oracle SQL/Services Result	Comments
Other transports All platforms	NULL	0	API allocates 1300	1300 is default

(continued on next page)

Table 6–4 (Cont.) Valid Combinations of Buffer-Related Parameters for the sqlsrv_associate Routine

Transport/Platform	Buffer Specified	Buffer Size Specified	Oracle SQL/Services Result	Comments
Other transports All platforms	NULL	256+	API allocates what user specified up to 32000	Client drops back to the server-supported value[1]
Other transports All platforms	Valid user-allocated buffer	256+	API uses what user specified up to 32000	Client drops back to the server-supported value[1]
IPX/SPX transport Windows enhanced mode	NULL	0	API allocates 534	534 is the default
IPX/SPX transport Windows enhanced mode	NULL	256+	API allocates up to 534	No error if specified value > 534
IPX/SPX transport Windows enhanced mode	Valid user-allocated buffer	256+	API uses up to 534 of user buffer	No error if specified value > 534
IPX/SPX transport Windows standard mode	NULL	0	API allocates 470	470 is the default
IPX/SPX transport Windows standard mode	NULL	256+	API allocates up to 470	No error if specified value > 470

[1] V7.0 server—client drops back to server-specified value; V6.1 or V6.0 server—client drops back to 5000.

(continued on next page)

Table 6–4 (Cont.) Valid Combinations of Buffer-Related Parameters for the sqlsrv_associate Routine

Transport/Platform	Buffer Specified	Buffer Size Specified	Oracle SQL/Services Result	Comments
IPX/SPX transport Windows standard mode	Valid user-allocated buffer	256+	API uses up to 470 of user buffer	No error if specified value > 470

write_buffer_size (optional)

The size in bytes of the API buffer used to send messages. If a write_buffer is passed, the write_buffer_size must contain its size. The minimum value is 256 bytes. If no write buffer is passed, Oracle SQL/Services allocates a buffer of size write-buffer-size if the parameter is non-zero, or of a default size if the parameter is zero. See Table 6–4 for valid combinations of buffer-related parameters. The values for write_buffer_size and read_buffer_size must be equal.

sqlca_str (optional)

Address of an SQLCA (SQL Communications Area) structure (see Section 7.3). If you specify a buffer address of NULL, Oracle SQL/Services allocates the SQLCA structure. Oracle Corporation recommends that you pass a NULL value. You must pass a NULL value when writing client applications on a Macintosh system.

The SQLCA structure is defined in the include file sqlsrvca.h. All valid error codes are defined in sqlsrv.h.

associate_str

Address of an ASSOCIATE_STR structure used to define optional association characteristics (see Section 7.2). The ASSOCIATE_STR structure is defined in the include file sqlsrv.h.

associate_id

A pointer to a variable of type ASSOCIATE_ID into which the API writes the association identifier. This identifier distinguishes one active association from all others.

Notes

- Errors that are detected early in the processing of the sqlsrv_associate routine are returned only in the longword return value from sqlsrv_associate. These errors include SQLSRV_INVARG, SQLSRV_INVSQLCA, SQLSRV_NO_MEM, and SQLSRV_INVBUFSIZ.
- If the read or write buffer size is less than 256 bytes, Oracle SQL/Services returns an SQLSRV_INVARG error on sqlsrv_associate.
- If the read_buffer or write_buffer parameter values are user-allocated buffers, but the read_buffer_size or write_buffer_size parameter values are specified as 0, Oracle SQL/Services returns an SQLSRV_INVARG error on sqlsrv_associate.
- If the read_buffer and write_buffer size are not of equal size, Oracle SQL/Services returns an SQLSRV_INVBUFSIZ error on sqlsrv_associate.
- When errors are detected before an associate_id is allocated for the associate session, the sqlsrv_associate routine writes NULL to the associate_id variable to indicate that no associate_id is assigned to this association. In this case, applications should not make subsequent Oracle SQL/Services API calls that require an associate_id.
- When errors are detected after an associate_id is allocated for the association, the sqlsrv_associate routine writes a non-NULL value to the associate_id variable. In this case, applications can make calls to a limited subset of Oracle SQL/Services API routines, such as the sqlsrv_sqlca_error and sqlsrv_sqlca_error_text routines, to retrieve additional information about the error. In this situation, the application should call the sqlsrv_release API routine to release the resources held by the association after retrieving the additional error information.
- If a client application connects to a V7.0 server using read and write buffer sizes that are larger than the server can handle, the sqlsrv_associate routine adjusts the buffer sizes locally and immediately returns a success status to the client application.

  If a client application connects to a V6.1 or V6.0 server using read and write buffer sizes that are larger than the server can handle, the sqlsrv_associate routine releases the network connection, then reconnects with both read and write buffer sizes set to 5000. When the network connection is reestablished, the sqlsrv_associate routine returns a success status to the client application.

In both situations, the mechanism used by the sqlsrv_associate routine to select an appropriate buffer size is transparent to the client application. Client applications can call the sqlsrv_get_associate_info routine to determine the actual buffer size being used for the association.

- The Oracle SQL/Services NetWare client uses server names in place of node names. A server name is passed to Oracle SQL/Services using the sqlsrv_associate routine in place of the node_name parameter. The maximum length of the server name is 48 bytes, as specified by Novell. Oracle SQL/Services requires this server name to be the name of the InterConnections File Server, which must be installed and running on the same node as the Oracle SQL/Services server. Oracle SQL/Services NetWare clients use the InterConnections File Server as a name server.
- When an association is no longer required, your application calls the sqlsrv_release routine to commit any outstanding transactions, release any prepared statements, disconnect the network link, and release any memory allocated to the association at the client and server.

Errors

SQLSRV_CONNTIMEOUT	The connection to the server could not be completed within the specified time limit.
SQLSRV_DLL_ADDR_ERR	Windows application GetProcAddress call error.
SQLSRV_DLL_LOAD_ERR	Windows application LoadLibrary call error.
SQLSRV_EXEINTERR	The executor has encountered an internal or other error condition.
SQLSRV_GETACCINF	Client authentication or authorization failed.
SQLSRV_HOSTERR	An attempt to access TCP/IP host files failed.
SQLSRV_INTERR	Internal error.
SQLSRV_INV_CLS	Invalid or unknown service name specified.
SQLSRV_INVARG	Invalid routine parameter.
SQLSRV_INVASCSTR	Invalid parameter in ASSOCIATE_STR.
SQLSRV_INVBUFSIZ	Invalid read or write buffer size.
SQLSRV_INVSQLCA	Invalid SQLCA structure.
SQLSRV_NETERR	Network transport returned an error.

SQLSRV_NO_CONNREC	Attempt to get a connection record from the Macintosh Communications ToolBox failed; check the node name.
SQLSRV_NO_MEM	API memory allocation failed.
SQLSRV_NO_PRCAVL	No executor processes are available to service the connection.
SQLSRV_NO_RSRC	Unable to open the Macintosh Control Panel Device resource.
SQLSRV_NO_SYSFLDR	Unable to open the Macintosh Control Panel Device in the System Folder.
SQLSRV_OPNLOGFIL	Unable to open log file.
SQLSRV_PWD_EXPIRED	The password has expired.
SQLSRV_SQLNET_BADCONNECT	SQL*Net is unable to connect to the server.
SQLSRV_SQLNET_BADINIT	Unable to initialize SQL*Net.
SQLSRV_SQLNET_BADSERVICE	SQL*Net is unable to resolve the service name being specified.
SQLSRV_SVCNOTRUN	The specified service is not running.
SQLSRV_SVC_SHUTDOWN	The specified service has been shut down.
SQLSRV_TOOMANYCONNECTS	The maximum number of network connections has been reached at the server.
SQLSRV_XPT_MISSING	The specified network transport is not installed or is not available on the client node operating system.

sqlsrv_get_associate_info

The sqlsrv_get_associate_info routine returns attributes of the association structure. The information is copied to a user buffer when sqlsrv_get_associate_info is called.

C Format

```
extern int sqlsrv_get_associate_info(
               ASSOCIATE_ID associate_id,
               unsigned short int info_type,
               unsigned short int buf_len,
               char *info_buf,
               SQS_LONGWORD *info_num);
```

Parameters

associate_id

An identifier used to distinguish one active association from all others.

info_type

Specifies the type of information to be returned. The values of the info_type parameter are shown in Table 6–5.

Table 6–5 Values of the info_type Parameter

Value	Description
SQLSRV_INFO_SQL_VERSION	Gets the version of SQL used by the server and returns it as character data.
SQLSRV_INFO_ENGINE	Gets the version of the Oracle Rdb database engine (if Oracle Rdb is used) used by the server and returns it as character data.
SQLSRV_INFO_SRV_VERSION	Gets the version of the Oracle SQL/Services server and returns it as character data.

(continued on next page)

Table 6–5 (Cont.) Values of the info_type Parameter

Value	Description
SQLSRV_INFO_PROTOCOL	Gets the protocol level of the server and returns it as a longword.
SQLSRV_INFO_SERVER_PID	Gets the process ID (PID) of the executor and returns it as a longword.
SQLSRV_INFO_TRANSPORT	Gets the transport type in use and returns the information as character data.
SQLSRV_INFO_BUFFER_SIZE	Gets the negotiated client buffer size and returns the information as a longword.
SQLSRV_INFO_SERVICE_ATTRS	Gets the service attributes and returns the value as a bit mask in a 32-bit longword. The bit mask is defined in Table 6–6.

The values of the SQLSRV_INFO_SERVICE_ATTRS bit masks are shown in Table 6–6.

Table 6–6 Values of the SQLSRV_INFO_SERVICE_ATTRS Bit Masks

Value	Numeric Value	Description
SQLSRV_INFO_SVC_DBSERVICE	1	Set if the service is a database service.
SQLSRV_INFO_SVC_REUSETXN	2	Set if the service is transaction reusable.
SQLSRV_INFO_SVC_TIEDEXEC	4	Set if the service is transaction reusable and the association is tied to a single executor for the life of the connection. For V7.0, this bit will always be set if the SQLSRV_INFO_SVC_REUSETXN bit is set.

buf_len

The size of a user-supplied buffer for information returned as character data.

info_buf

Address of a user-supplied buffer of type char for information returned as character data. This is required for information returned as character data.

info_num

The address of a variable of type SQS_LONGWORD to be used for information returned as a longword, or for the number of characters returned for information returned as character data. This is required for information returned as a longword, and optional for information returned as character data.

Notes

- The sqlsrv_get_associate_info service returns one attribute per call. To get multiple attributes, your application must call sqlsrv_get_associate_info once for each attribute.
- For information returned as character data, if the actual length of the string is longer than the user-supplied buffer, the returned information is truncated to the size of the buffer.

Errors

SQLSRV_INVARG	Invalid routine parameter.
SQLSRV_INVASC	Invalid association identifier.
SQLSRV_SRVNOTSUP	The server is not supported.

sqlsrv_release

The sqlsrv_release routine commits active transactions on the server and requests an orderly termination of the association, which disconnects the network link and frees client association resources.

C Format

```
extern int sqlsrv_release(
                ASSOCIATE_ID associate_id,
                char *stats);
```

Parameters

associate_id

An identifier used to distinguish one active association from all others.

stats (optional)

This parameter must be 0 or NULL.

Notes

- When an association is no longer required, your application calls the sqlsrv_release routine to commit any outstanding transactions, release any prepared statements, disconnect the network link, and release any memory allocated to the association at the client and server.

Errors

SQLSRV_CONNTIMEOUT	The connection to the server could not be completed within the specified time limit.
SQLSRV_EXEINTERR	The executor has encountered an internal or other error condition.
SQLSRV_INTERR	Internal error.
SQLSRV_INVASC	Invalid association identifier.
SQLSRV_MULTI_ACT	A batched sqlsrv_execute_in_out or sqlsrv_fetch_many context is active.

SQLSRV_NETERR	Network transport returned an error.
SQLSRV_SVC_SHUTDOWN	The specified service has been shut down.

6.3.2 SQL Statement Routines

SQL statement routines prepare and execute SQL statements, and release prepared SQL statement resources. These routines map to the dynamic SQL interface. SQL statement routines include the following routines:

- sqlsrv_prepare routine (see sqlsrv_prepare)
- sqlsrv_execute_in_out routine (see sqlsrv_execute_in_out)
- sqlsrv_execute_immediate routine (see sqlsrv_execute_immediate)
- sqlsrv_release_statement routine (see sqlsrv_release_statement)

sqlsrv_prepare

The sqlsrv_prepare routine prepares the input SQL statement and returns a value that identifies the prepared statement. It also optionally allocates and initializes SQLDA or SQLDA2 parameter markers and select list items associated with the SQL statement.

C Format

```
extern int sqlsrv_prepare(
                ASSOCIATE_ID associate_id,
                SQS_LONGWORD database_id,
                char *sql_statement,
                SQS_LONGWORD *statement_id,
                SQLDA_ID *parameter_marker_sqlda,
                SQLDA_ID *select_list_sqlda);
```

Parameters

associate_id

An identifier used to distinguish one active association from all others.

database_id

This parameter must be 0. Databases are referenced within the SQL statement syntax.

sql_statement

Address of a null-terminated string containing the SQL statement to be prepared.

statement_id

Address of a variable of type SQS_LONGWORD into which the API writes the identifier used in all subsequent references to the prepared statement.

parameter_marker_sqlda

A pointer to a variable of type SQLDA_ID.

Oracle Corporation recommends that you let the Oracle SQL/Services client API allocate memory for each parameter marker SQLDA or SQLDA2, in which case your application should store NULL in the parameter marker SQLDA_ID before calling sqlsrv_prepare. If your application provides its own memory for each parameter marker SQLDA or SQLDA2, your application must store the address of that memory in the parameter marker SQLDA_ID before calling sqlsrv_prepare.

If the SQL statement is prepared successfully, Oracle SQL/Services allocates memory for the SQLDA or SQLDA2, stores the address in the SQLDA_ID, if necessary, and writes metadata information about all the parameter markers contained in the SQL statement to the parameter marker SQLDA or SQLDA2.

select_list_sqlda

A pointer to a variable of type SQLDA_ID.

Oracle Corporation recommends that you let the Oracle SQL/Services client API allocate memory for each select list SQLDA or SQLDA2, in which case your application should store NULL in the select list SQLDA_ID before calling sqlsrv_prepare. If your application provides its own memory for each select list SQLDA or SQLDA2, your application must store the address of that memory in the select list SQLDA_ID before calling sqlsrv_prepare.

If the SQL statement is prepared successfully, Oracle SQL/Services allocates memory for the SQLDA or SQLDA2, stores the address in the SQLDA_ID, if necessary, and writes metadata information about all the select list items contained in the SQL statement to the select list SQLDA or SQLDA2.

Notes

- Oracle Corporation recommends that you let the Oracle SQL/Services client API allocate memory for each parameter marker and select list SQLDA or SQLDA2. To check for the presence of parameter markers or select list items in this situation, your application tests the respective SQLDA_ID for a non-NULL value. If the SQLDA_ID does contain a non-NULL value, the number of parameter markers or select list items may be obtained from the SQLD field of the SQLDA or SQLDA2 using the sqlsrv_sqlda_sqld or sqlsrv_sqlda2_sqld routines.
- If your application provides it own memory for each parameter marker and select list SQLDA or SQLDA2, it must initialize the SQLDAID field to "SQLDA" or "SQLDA2"; the SQLDABC field to the total size, in bytes, of the SQLDA; the SQLD field to zero; and the SQLN field to the total number of SQLVARs or SQLVAR2s in the SQLDA or SQLDA2. Upon successful completion of a call to sqlsrv_prepare, the presence and number

of parameter markers or select list items is indicated by a non-zero value in the SQLD field of the SQLDA or SQLDA2.

Note

On the Macintosh operating system, your application cannot provide its own memory for SQLDAs or SQLDA2s, but must let the Oracle SQL/Services client API allocate the memory for SQLDAs and SQLDA2s.

- To enable your application to distinguish between different types of SQL statements, Oracle Rdb stores the statement type in the SQLERRD[1] field of the SQLCA. The statement types, as defined by Oracle Rdb, are as follows:

  0: statement is an executable statement other than a CALL statement
  1: statement is a SELECT statement
  2: statement is a CALL statement

  You can retrieve this value using the sqlsrv_sqlca_sqlerrd routine.
- If the prepared statement is a CALL statement, the metadata for any input or input/output arguments is written to the parameter marker SQLDA or SQLDA2, while the metadata for any output or input/output arguments is written to the select list SQLDA or SQLDA2. Note that metadata for each input/output argument is written to both the parameter marker and select list SQLDAs or SQLDA2s. However, in all other respects, your application processes a CALL statement in the same manner as any other executable SQL statement.

Errors

SQLSRV_CONNTIMEOUT	The connection to the server could not be completed within the specified time limit.
SQLSRV_EXEINTERR	The executor has encountered an internal or other error condition.
SQLSRV_INTERR	Internal error.
SQLSRV_INVARG	Invalid routine parameter.
SQLSRV_INVASC	Invalid association identifier.
SQLSRV_INVSQLDA	Invalid SQLDA, SQLDA2, or SQLDA_ID.

SQLSRV_NETERR	Network transport returned an error.
SQLSRV_NO_MEM	API memory allocation failed.
SQLSRV_SVC_SHUTDOWN	The specified service has been shut down.

sqlsrv_execute_in_out

The sqlsrv_execute_in_out routine executes any prepared, executable SQL statement. The prepared statement may accept input from a parameter marker SQLDA or SQLDA2, or return output in a select list SQLDA or SQLDA2, or both. The sqlsrv_execute_in_out routine supersedes the sqlsrv_execute routine.

C Format

```
extern int sqlsrv_execute_in_out(
                ASSOCIATE_ID associate_id,
                SQS_LONGWORD database_id,
                SQS_LONGWORD statement_id,
                short int execute_flag,
                SQLDA_ID parameter_marker_sqlda,
                SQLDA_ID select_list_sqlda);
```

Parameters

associate_id

An identifier used to distinguish one active association from all others.

database_id

This parameter must be 0. Databases are referenced within the SQL statement syntax.

statement_id

The statement ID returned previously by sqlsrv_prepare when the statement was prepared. If you start batched execution for a particular statement ID using the SQLSRV_EXE_BATCH flag, you must end batched execution for that statement ID using one of the SQLSRV_EXE_W_DATA, SQLSRV_EXE_WO_DATA, or SQLSRV_EXE_ABORT flags before you can execute any other prepared statement.

execute_flag

For a prepared statement that contains parameter markers, this parameter specifies whether the API sends single or multiple sets of parameter marker values to the server for processing (see Section 4.1 for more information on batched execution). For all other prepared SQL statements, this value must be 0 (SQLSRV_EXE_W_DATA). The values of the execute_flag parameter are shown in Table 6–7.

Table 6–7 Values of the execute_flag Parameter in sqlsrv_execute_in_out

Flag Name	Value	Description
SQLSRV_EXE_W_DATA	0	Builds an execute request message in the message buffer using the current values in the parameter marker SQLDA or SQLDA2, then sends the message to the server for execution. If batched execution is currently in effect for the statement, this parameter appends the new message to the previous messages in the message buffer, and sends all the messages to the server for execution along with any requests already queued at the server.
SQLSRV_EXE_BATCH	1	Starts or continues batched execution by building an execute request message in the message buffer using the current values in the parameter marker SQLDA or SQLDA2. If batched execution is already in effect for the statement, this parameter appends the new message to the previous messages in the message buffer. Using batched execution, no messages are sent to the server until the message buffer fills up, whereupon the messages in the message buffer are sent to the server to be queued up for subsequent execution behind any previously queued requests.

(continued on next page)

Table 6–7 (Cont.) Values of the execute_flag Parameter in sqlsrv_execute_in_out

Flag Name	Value	Description
SQLSRV_EXE_WO_DATA	2	Ends batched execution by sending the current contents of the message buffer to the server for execution along with any previously queued requests. Note that the current values in the parameter marker SQLDA or SQLDA2 are *not* sent to the server when batched execution is ended using the SQLSRV_EXE_WO_DATA flag.
SQLSRV_EXE_ABORT	3	Aborts batched execution by discarding the current contents of the message buffer and sending a message to the server directing it to discard any previously queued requests.

parameter_marker_sqlda

An SQLDA_ID that identifies the parameter marker SQLDA or SQLDA2 containing any parameter marker values or input and input/output argument values for the SQL statement to be executed.

select_list_sqlda

An SQLDA_ID that identifies the select list SQLDA or SQLDA2 to receive any select list items or output and input/output argument values returned by the SQL statement to be executed.

Notes

- On successful completion of a call to sqlsrv_execute_in_out, Oracle SQL/Services stores the total number of database rows inserted, updated, or deleted in the SQLERRD[2] field of the SQLCA. Because multiple rows may be updated or deleted when you execute an UPDATE or DELETE statement, this value may be higher than the number of times that you called sqlsrv_execute_in_out for a particular batched execution. You can retrieve the row count from the SQLCA using the sqlsrv_sqlca_count routine. Note that Oracle Rdb does not return a row count value if you use the CALL statement to invoke a stored procedure, or if you execute a compound statement.

- If an error occurs executing a request queued for batched execution, then the server discards any remaining requests in the batch execution queue and returns the error to the client. Currently, there is no way to determine precisely which request caused the failure. Therefore, client applications will typically roll back the transaction in this situation.
- If you use batched execution to execute an SQL statement containing both parameter markers and select list items, such as UPDATE . . . RETURNING, then only the results from the execution of the last queued request are returned to the client. The results from the execution of all previously queued requests are lost.
- Once you start batched execution for a particular statement ID, you cannot call any API routines other than sqlsrv_execute_in_out, nor can you execute any other prepared statements until you end batched execution for the current statement ID using one of the SQLSRV_EXE_W_DATA, SQLSRV_EXE_WO_DATA, or SQLSRV_EXE_ABORT flags.
- SQL describes the metadata for any items specified in the RETURNING clause of an INSERT statement into the end of the parameter marker SQLDA or SQLDA2. Note that columns, output arguments, and other values returned by a statement are normally described in the select list SQLDA or SQLDA2. The server does not normally return data values from a parameter marker SQLDA or SQLDA2 to the client; therefore, the server must explicitly check each parameter marker SQLDA or SQLDA2 to determine if an INSERT statement contains a RETURNING clause. To do so, it checks the name of the last column described in the parameter marker SQLDA or SQLDA2 for the value DBKEY. Therefore, the only value that can be returned from an INSERT statement is the DBKEY, because the server is unable to detect any other returned value. For example:

  ```
  INSERT INTO EMPLOYEES VALUES ( ?,?,?,?,?,?,?,? ) RETURNING DBKEY INTO ?;
  ```

  SQL describes the metadata for any items specified in the RETURNING clause of an UPDATE statement into the select list SQLDA or SQLDA2 as expected.

Errors

SQLSRV_CONNTIMEOUT	The connection to the server could not be completed within the specified time limit.
SQLSRV_EXEINTERR	The executor has encountered an internal or other error condition.
SQLSRV_DATA_TOO_LONG	The Oracle SQL/Services executor determined that the length of a data value in an SQLDA exceeded the maximum allowed for the value's data type.
SQLSRV_INTERR	Internal error.
SQLSRV_INVARG	Invalid routine parameter.
SQLSRV_INVASC	Invalid association identifier.
SQLSRV_INVEXEFLG	Invalid execute flag.
SQLSRV_INVSELLST	Invalid SQLDA or SQLDA2 select list.
SQLSRV_INVSQLDA	Invalid SQLDA, SQLDA2, or SQLDA_ID.
SQLSRV_INVSTMID	Invalid statement identifier.
SQLSRV_MULTI_ACT	A batched sqlsrv_execute_in_out or sqlsrv_fetch_many context is active.
SQLSRV_NETERR	Network transport returned an error.
SQLSRV_SVC_SHUTDOWN	The specified service has been shut down.

sqlsrv_execute_immediate

The sqlsrv_execute_immediate routine prepares and executes an SQL statement that does not contain parameter markers or select list items.

C Format

```
extern int sqlsrv_execute_immediate(
                ASSOCIATE_ID associate_id,
                SQS_LONGWORD database_id,
                char *sql_statement);
```

Parameters

associate_id

An identifier used to distinguish one active association from all others.

database_id

This parameter must be 0. Databases are referenced within the SQL statement syntax.

sql_statement

Address of a null-terminated string containing the SQL statement to be prepared and executed by dynamic SQL.

Notes

- sqlsrv_execute_immediate provides an efficient mechanism, using a single request/response message pair, for executing an SQL statement that does not contain any parameter markers or select list items where the statement is to be executed only once. However, if the same SQL statement is to be executed multiple times, it is more efficient to prepare the statement and execute it as necessary, even if the statement contains no parameter markers or select list items.
- On successful completion of a call to sqlsrv_execute_immediate, Oracle SQL/Services stores the total number of database rows updated or deleted in the SQLERRD[2] field of the SQLCA. You can retrieve the row count from the SQLCA using the sqlsrv_sqlca_count routine. Note that Oracle Rdb does not return a row count value if you use the CALL statement to invoke a stored procedure, or if you execute a compound statement.

Errors

SQLSRV_CONNTIMEOUT	The connection to the server could not be completed within the specified time limit.
SQLSRV_EXEINTERR	The executor has encountered an internal or other error condition.
SQLSRV_INTERR	Internal error.
SQLSRV_INVARG	Invalid routine parameter.
SQLSRV_INVASC	Invalid association identifier.
SQLSRV_MULTI_ACT	A batched sqlsrv_execute_in_out or sqlsrv_fetch_many context is active.
SQLSRV_NETERR	Network transport returned an error.
SQLSRV_SVC_SHUTDOWN	The specified service has been shut down.

sqlsrv_release_statement

The sqlsrv_release_statement routine frees all resources associated with one or more prepared statements at both the client and server. The sqlsrv_release_statement routine implicitly invokes sqlsrv_free_sqlda_data or sqlsrv_free_sqlda2_data to free dynamically allocated SQLDA or SQLDA2 structures.

C Format

```
extern int sqlsrv_release_statement(
                ASSOCIATE_ID associate_id,
                short int statement_id_count,
                SQS_LONGWORD *statement_id_array);
```

Parameters

associate_id

An identifier used to distinguish one active association from all others.

statement_id_count

The number of statement identifiers passed in the statement_id_array.

statement_id_array

An array containing the identifiers (statement_id parameters returned by the sqlsrv_prepare routine) of the statements to free.

Notes

- You cannot release a statement that has an open cursor.
- If you call sqlsrv_allocate_sqlda_data or sqlsrv_allocate_sqlda2_data to allocate memory for parameter marker and select list item data and indicator variables, Oracle SQL/Services automatically frees the memory when you call sqlsrv_release_statement. If you let sqlsrv_prepare allocate memory for the parameter marker and select list SQLDA or SQLDA2 structures, Oracle SQL/Services automatically frees the memory when you call sqlsrv_release_statement.

- If Oracle SQL/Services encounters an error validating or releasing a particular statement ID, it discards any subsequent statement IDs and returns the error to the client application. Oracle SQL/Services stores the total number of statements released in the SQLERRD[2] field of the SQLCA. You can retrieve the count from the SQLCA using the sqlsrv_ sqlca_count routine.

Errors

SQLSRV_CONNTIMEOUT	The connection to the server could not be completed within the specified time limit.
SQLSRV_EXEINTERR	The executor has encountered an internal or other error condition.
SQLSRV_INTERR	Internal error.
SQLSRV_INVARG	Invalid routine parameter.
SQLSRV_INVASC	Invalid association identifier.
SQLSRV_INVSTMID	Invalid statement identifier.
SQLSRV_MULTI_ACT	A batched sqlsrv_execute_in_out or sqlsrv_ fetch_many context is active.
SQLSRV_NETERR	Network transport returned an error.
SQLSRV_SVC_SHUTDOWN	The specified service has been shut down.

6.3.3 Result Table Routines

Result table routines allow the caller to fetch data from the server by providing calls to open a cursor, fetch from an open cursor, and close an open cursor. Result table routines include the following routines:

- sqlsrv_declare_cursor routine (see sqlsrv_declare_cursor)
- sqlsrv_open_cursor routine (see sqlsrv_open_cursor)
- sqlsrv_fetch routine (see sqlsrv_fetch)
- sqlsrv_fetch_many routine (see sqlsrv_fetch_many)
- sqlsrv_close_cursor routine (see sqlsrv_close_cursor)

sqlsrv_declare_cursor

The sqlsrv_declare_cursor routine declares a dynamic cursor. If you do not use the sqlsrv_declare_cursor routine, Oracle SQL/Services implicitly declares all cursors as type table and mode update within the sqlsrv_open_cursor call.

C Format

```
extern int sqlsrv_declare_cursor(
                ASSOCIATE_ID associate_id,
                char *cursor_name
                SQS_LONGWORD statement_id
                short int cursor_type
                short int cursor_mode);
```

Parameters

associate_id

An identifier used to distinguish one active association from all others.

cursor_name

Address of a null-terminated string used to identify the cursor.

statement_id

The statement ID returned previously by sqlsrv_prepare when the SELECT statement was prepared. The sqlsrv_declare_cursor routine maps the cursor_name to the prepared statement.

cursor_type

A value indicating the type of list cursor to declare. You can declare table or list cursors:

- Table

  Declare table cursors by specifying the SQLSRV_TABLE_CURSOR literal.

- List

  Declare list cursors by specifying the SQLSRV_LIST_CURSOR literal.

For detailed information about SQL list and table cursors, refer to the *Oracle Rdb7 Guide to SQL Programming* and the *Oracle Rdb7 SQL Reference Manual*.

cursor_mode

A value indicating the mode of table or list cursors. Table cursors have four modes:

- Update-only

  To declare table cursors in update-only mode, specify the literal SQLSRV_MODE_UPDATE_ONLY.
- Update

  To declare table cursors in update mode, specify the literal SQLSRV_MODE_UPDATE.
- Read-only

  To declare table cursors in read-only mode, specify the literal SQLSRV_MODE_READ_ONLY.
- Insert-only

  To declare table cursors in insert-only mode, specify the literal SQLSRV_MODE_INSERT_ONLY.

List cursors have three modes:

- Read-only

  To declare list cursors in read-only mode, specify the literal SQLSRV_MODE_READ_ONLY.
- Insert-only

  To declare list cursors in insert-only mode, specify the literal SQLSRV_MODE_INSERT_ONLY.
- Scroll

  To declare list cursors in scroll mode, specify the literal SQLSRV_MODE_SCROLL.

For detailed information about SQL cursor modes, refer to the *Oracle Rdb7 Guide to SQL Programming* and the *Oracle Rdb7 SQL Reference Manual*.

Notes

- When designing applications, you should avoid using cursor names starting with the prefix "SQLSRV_"; this is a reserved prefix and is used by the Oracle SQL/Services product.
- The cursor type and cursor mode literals are defined in the sqlsrv.h file.

Errors

SQLSRV_DUPCRSNAM	Duplicate cursor name.
SQLSRV_INTERR	Internal error.
SQLSRV_INVARG	Invalid routine parameter.
SQLSRV_INVASC	Invalid association identifier.
SQLSRV_INVCURNAM	Invalid cursor name.
SQLSRV_INVSTMID	Invalid statement identifier.
SQLSRV_MULTI_ACT	A batched sqlsrv_execute_in_out or sqlsrv_fetch_many context is active.
SQLSRV_NETERR	Network transport returned an error.

sqlsrv_open_cursor

The sqlsrv_open_cursor routine opens a cursor for a prepared SELECT statement. The sqlsrv_declare_cursor routine optionally determines the type and mode of the cursor.

C Format

```
extern int sqlsrv_open_cursor(
                ASSOCIATE_ID associate_id,
                char *cursor_name,
                SQS_LONGWORD statement_id,
                SQLDA_ID parameter_marker_sqlda);
```

Parameters

associate_id

An identifier used to distinguish one active association from all others.

cursor_name

Address of a null-terminated string identifying the cursor. All cursor operations, including positional INSERT, UPDATE, and DELETE statements, must use the cursor name to identify the cursor.

statement_id

The statement ID returned previously by sqlsrv_prepare when the SELECT statement was prepared. The sqlsrv_open_cursor routine maps the cursor_name to the prepared statement.

parameter_marker_sqlda

An SQLDA identifier defining the parameter marker values for the prepared SELECT statement.

Notes

- After a successful call to sqlsrv_open_cursor to open a table cursor, Oracle Rdb stores the following information in the SQLCA:
  - Estimated result table cardinality in the SQLERRD[2] field.
  - Estimated I/Os in the SQLERRD[3] field.

  These values are retrieved using the sqlsrv_sqlca_sqlerrd routine.
- After a successful call to sqlsrv_open_cursor to open a list cursor, Oracle Rdb stores the following information in the SQLCA:
  - Length of the largest actual segment in the SQLERRD[1] field.
  - Total number of segments in the SQLERRD[3] field.
  - Total length of all the segments as a quadword value in the SQLERRD[4] and SQLERRD[5] fields, which contain the low-order 32 bits and high-order 32 bits, respectively.

  These values are retrieved using the sqlsrv_sqlca_sqlerrd routine.

Errors

SQLSRV_CONNTIMEOUT	The connection to the server could not be completed within the specified time limit.
SQLSRV_DATA_TOO_LONG	The Oracle SQL/Services executor determined that the length of a data value in an SQLDA exceeded the maximum allowed for the value's data type.
SQLSRV_EXEINTERR	The executor has encountered an internal or other error condition.
SQLSRV_INTERR	Internal error.
SQLSRV_INVARG	Invalid routine parameter.
SQLSRV_INVASC	Invalid association identifier.
SQLSRV_INVCURNAM	Invalid cursor name.
SQLSRV_INVSQLDA	Invalid SQLDA, SQLDA2, or SQLDA_ID.
SQLSRV_INVSTMID	Invalid statement identifier.
SQLSRV_MULTI_ACT	A batched sqlsrv_execute_in_out or sqlsrv_fetch_many context is active.
SQLSRV_NETERR	Network transport returned an error.

SQLSRV_SVC_SHUTDOWN	The specified service has been shut down.

sqlsrv_fetch

The sqlsrv_fetch routine fetches a row of data into a select list SQLDA.

C Format

```
extern int sqlsrv_fetch(
                ASSOCIATE_ID associate_id,
                char *cursor_name,
                short int scroll_option,
                SQS_LONGWORD position,
                SQLDA_ID select_list_sqlda);
```

Parameters

associate_id

An identifier used to distinguish one active association from all others.

cursor_name

Address of a null-terminated string used to identify the open cursor.

scroll_option

The values of the scroll_option parameter are shown in Table 6–8.

Table 6–8 Values of the scroll_option Parameter

Value	Description
SQLSRV_NO_SCROLL	No scroll option.
SQLSRV_SCROLL_FIRST	Fetch first segment.
SQLSRV_SCROLL_LAST	Fetch last segment.
SQLSRV_SCROLL_PRIOR	Fetch prior segment.
SQLSRV_SCROLL_NEXT	Fetch next segment.
SQLSRV_SCROLL_ABSOLUTE	Fetch an absolute segment of the list cursor.

(continued on next page)

Table 6–8 (Cont.) Values of the scroll_option Parameter

Value	Description
SQLSRV_SCROLL_RELATIVE	Fetch a relative segment relative to the current list cursor position.

For table cursors, the scroll option must be 0 (SQLSRV_NO_SCROLL). For scrollable list cursors, a value of SQLSRV_SCROLL_ABSOLUTE indicates an absolute segment within the segmented string, while a value of SQLSRV_ SCROLL_RELATIVE indicates a segment relative to the current cursor position. When a parameter value of SQLSRV_SCROLL_ABSOLUTE or SQLSRV_SCROLL_RELATIVE is specified, the value specified for the position argument indicates the position value.

position

Indicates the position value for an absolute or relative scroll option. For an absolute scroll option, this parameter value indicates the *n*th absolute list segment of the list cursor. For a relative scroll option, this parameter value (positive or negative) indicates the *n*th list segment relative to the current list cursor position. For example, a value of –5 for the position parameter for a relative scroll option results in a fetch of the 5th segment previous to the current cursor position. The position parameter value must be 0 if the scroll_option parameter is not a relative or absolute scroll option.

select_list_sqlda

The select list SQLDA identifier in which to store the row.

Notes

- A return value of SQL_EOS indicates end of data, that is, the result table is empty, or no more rows remain in the result table. A call to the sqlsrv_ fetch routine that returns a status code of SQL_EOS does not return any data in the SQLDA.
- Although it returns only one row to the application for each call, the sqlsrv_fetch routine can request that the server send multiple rows of data from the server when called within an sqlsrv_fetch_many context. See Fetching Multiple Rows in Chapter 4 and sqlsrv_fetch_many.
- To scroll read-only list cursors, the scroll_option argument must specify a value as indicated in Table 6–8, and the position argument must specify the position value when an absolute or relative scroll_option value is specified. Otherwise, the position argument must be 0.

- After a successful call to sqlsrv_fetch, Oracle SQL/Services stores the number of the current row within the result table in the SQLERRD[2] field of the SQLCA. This value can be retrieved using the sqlsrv_sqlca_sqlerrd routine.

Errors

SQLSRV_CONNTIMEOUT	The connection to the server could not be completed within the specified time limit.
SQLSRV_EXEINTERR	The executor has encountered an internal or other error condition.
SQLSRV_INTERR	Internal error.
SQLSRV_INVARG	Invalid routine parameter.
SQLSRV_INVASC	Invalid association identifier.
SQLSRV_INVCURNAM	Invalid cursor name.
SQLSRV_INVSQLDA	Invalid SQLDA, SQLDA2, or SQLDA_ID.
SQLSRV_MULTI_ACT	A batched sqlsrv_execute_in_out or sqlsrv_fetch_many context is active.
SQLSRV_NETERR	Network transport returned an error.
SQLSRV_SVC_SHUTDOWN	The specified service has been shut down.

sqlsrv_fetch_many

The sqlsrv_fetch_many routine directs the sqlsrv_fetch routine to transfer multiple rows of data from the server, as described in Fetching Multiple Rows in Chapter 4. Frequently, this reduces the number of client/server messages required to retrieve data from the server. By default, sqlsrv_fetch retrieves one row of data at a time from the server.

C Format

```
extern int sqlsrv_fetch_many(
                ASSOCIATE_ID associate_id,
                char *cursor_name,
                short int increment,
                short int repeat_count);
```

Parameters

associate_id

An identifier used to distinguish one active association from all others.

cursor_name

Address of a null-terminated string used to identify the open cursor.

increment

For a scrollable list cursor, the client API implicitly enables relative scroll mode (SQLSRV_SCROLL_RELATIVE) to fetch segments and uses the value in the increment argument to specify the relative position. Therefore, to fetch all segments in a segmented string, specify an increment value of 1. See sqlsrv_fetch for more information on scroll modes and relative positions. This argument is ignored for cursors other than scrollable list cursors.

repeat_count

The number of rows to fetch. A value of 0 fetches the entire result table. A value other than 0 fetches that number of rows. For example, an application might fetch enough rows to fill one screen.

Notes

- To achieve the best performance, Oracle Corporation recommends that you specify a repeat_count of 0 to fetch all records.
- When you specify a repeat_count other than 0, your application must call the sqlsrv_fetch_many routine again once the specified number of rows have been fetched. Otherwise, the API returns to the default behavior (one row for each call to the sqlsrv_fetch routine). See Fetching Multiple Rows in Chapter 4 for more information.
- Once you initiate an sqlsrv_fetch_many operation, you must fetch the specified number of rows using sqlsrv_fetch or close the cursor using sqlsrv_close_cursor before you call other API routines. You can call sqlsrv_close_cursor at any time to close the cursor and end the sqlsrv_fetch_many operation before all the rows have been fetched. Otherwise, you must call sqlsrv_fetch the necessary number of times to fetch all the rows from the result table if you specify a repeat count of zero or the specified number of rows if you specify a non-zero repeat count before you can call any other API routine.
- A call to the sqlsrv_close_cursor routine completes an sqlsrv_fetch_many operation.
- By default, the sqlsrv_fetch routine fetches only one row of data from the server. That way, your application can execute SQL statements INSERT . . . WHERE CURRENT OF cursor-name, UPDATE . . . WHERE CURRENT OF cursor-name, and DELETE . . . WHERE CURRENT OF cursor-name.
- The sqlsrv_fetch_many routine initiates an sqlsrv_fetch_many operation; however, it does not fetch any rows. Therefore, sqlsrv_fetch_many returns a success status even if there are no rows in the result table. In this situation, sqlsrv_fetch returns a status of SQL_EOS the first time it is called to fetch a row from the result table.
- The Oracle SQL/Services NetWare client does not support fetching multiple rows using sqlsrv_fetch_many. The sqlsrv_fetch_many routine, when used with the IPX/SPX transport, always returns a success status, but does not start an sqlsrv_fetch_many operation. Therefore, existing applications may call sqlsrv_fetch_many, but will not see the performance improvements normally associated with this call. The Oracle SQL/Services NetWare client does not batch result tuples to reduce the number of Oracle SQL/Services messages, due to SPX flow control limitations.

sqlsrv_fetch_many

Errors

SQLSRV_FTCMNYACT	An sqlsrv_fetch_many context is already active for this cursor.
SQLSRV_INTERR	Internal error.
SQLSRV_INVARG	Invalid routine parameter.
SQLSRV_INVASC	Invalid association identifier.
SQLSRV_INVCURNAM	Invalid cursor name.
SQLSRV_MULTI_ACT	A batched sqlsrv_execute_in_out or sqlsrv_fetch_many context is active.

sqlsrv_close_cursor

The sqlsrv_close_cursor routine closes an open cursor.

C Format

```
extern int sqlsrv_close_cursor(
               ASSOCIATE_ID associate_id,
               char *cursor_name);
```

Parameters

associate_id

An identifier used to distinguish one active association from all others.

cursor_name

Address of a null-terminated string used to identify the open cursor.

Errors

SQLSRV_CONNTIMEOUT	The connection to the server could not be completed within the specified time limit.
SQLSRV_EXEINTERR	The executor has encountered an internal or other error condition.
SQLSRV_INTERR	Internal error.
SQLSRV_INVASC	Invalid association identifier.
SQLSRV_INVCURNAM	Invalid cursor name.
SQLSRV_NETERR	Network transport returned an error.
SQLSRV_SVC_SHUTDOWN	The specified service has been shut down.

6.3.4 Utility Routines

Utility routines provide local service to the caller. Utility routines include the following routines:

- sqlsrv_allocate_sqlda_data or sqlsrv_allocate_sqlda2_data routine (see sqlsrv_allocate_sqlda_data or sqlsrv_allocate_sqlda2_data)
- sqlsrv_free_sqlda_data or sqlsrv_free_sqlda2_data routine (see sqlsrv_free_sqlda_data or sqlsrv_free_sqlda2_data)
- sqlsrv_set_option routine (see sqlsrv_set_option)

sqlsrv_allocate_sqlda_data or sqlsrv_allocate_sqlda2_data

The sqlsrv_allocate_sqlda_data or sqlsrv_allocate_sqlda2_data routine dynamically allocates memory for data and indicator variables. Your application passes an SQLDA_ID identifier to sqlsrv_allocate_sqlda_data or sqlsrv_allocate_sqlda2_data, which allocates buffers of the appropriate size and writes the addresses of the newly allocated buffers into the SQLDATA and SQLIND fields in the SQLVAR or SQLVAR2 array.

Note

You must not modify the SQLDATA and SQLIND fields in the SQLVAR or SQLVAR2 fields if you call sqlsrv_allocate_sqlda_data or sqlsrv_allocate_sqlda2_data to allocate memory for data and indicator variables. The operation and results of other client API routines will be unpredictable if you modify these fields. The format, parameters, description, notes, and errors for the SQLDA or SQLDA2 routines are identical unless otherwise specified.

C Format

```
extern int sqlsrv_allocate_sqlda_data(
                ASSOCIATE_ID associate_id,
                SQLDA_ID sqlda_str);

extern int sqlsrv_allocate_sqlda2_data(
                ASSOCIATE_ID associate_id,
                SQLDA_ID sqlda_str);
```

Parameters

associate_id

An identifier used to distinguish one active association from all others.

sqlda_str

The identifier of a parameter marker or select list SQLDA or SQLDA2 for which to allocate data and indicator variables.

Notes

- You can free buffers allocated by the sqlsrv_allocate_sqlda_data or sqlsrv_allocate_sqlda2_data routine explicitly by calling the sqlsrv_free_sqlda_data or sqlsrv_free_sqlda2_data routine, or implicitly by calling the sqlsrv_release_statement or sqlsrv_release routine.
- The sqlsrv_allocate_sqlda_data or sqlsrv_allocate_sqlda2_data routine allocates additional memory for certain data types, as shown in Table 6–9.

Table 6–9 Special Requirements of Data Types to Determine Extra Byte Lengths to Allocate

Data Type	Extra Memory to Allocate
SQLSRV_ASCII_STRING	+1 for null-terminating select list item values; note that parameter marker values are not treated as null-terminated strings
SQLSRV_GENERALIZED_DATE	+1 for null terminator
SQLSRV_INTERVAL	+1 for null terminator
SQLSRV_GENERALIZED_NUMBER	+6 for null terminator and to allow input in scientific notation [for example, 9999E+123]
SQLSRV_VARCHAR	+2 for SQLDAs or +4 for SQLDA2s for leading length field
SQLSRV_VARBYTE	+2 for SQLDAs or +4 for SQLDA2s for leading length field

Errors

SQLSRV_INTERR	Internal error.
SQLSRV_INVARG	Invalid routine parameter.
SQLSRV_INVASC	Invalid association identifier.
SQLSRV_INVDATTYP	Invalid data type.
SQLSRV_INVSQLDA	Invalid SQLDA, SQLDA2, or SQLDA_ID.
SQLSRV_NO_MEM	API memory allocation failed.
SQLSRV_USRDATALL	The user, not Oracle SQL/Services, has allocated data buffers.

sqlsrv_free_sqlda_data or sqlsrv_free_sqlda2_data

The sqlsrv_free_sqlda_data or sqlsrv_free_sqlda2_data routine frees buffers that hold data and indicator variables that were dynamically allocated by the sqlsrv_allocate_sqlda_data or sqlsrv_allocate_sqlda2_data routine. Your application passes an SQLDA_ID identifier to the API, which frees the buffers and writes zeros into the SQLDATA and SQLIND fields of the SQLVAR or SQLVAR2 array.

Note

The sqlsrv_release_statement and sqlsrv_release routines implicitly call the sqlsrv_free_sqlda_data or sqlsrv_free_sqlda2_data routine for each prepared statement's dynamically allocated SQLDA or SQLDA2 structure. The format, parameters, description, notes, and errors for the SQLDA or SQLDA2 routines are identical unless otherwise specified.

C Format

```
extern int sqlsrv_free_sqlda_data(
                ASSOCIATE_ID associate_id,
                SQLDA_ID sqlda_str);

extern int sqlsrv_free_sqlda2_data(
                ASSOCIATE_ID associate_id,
                SQLDA_ID sqlda_str);
```

Parameters

associate_id

An identifier used to distinguish one active association from all others.

sqlda_str

The identifier of a parameter marker or select list SQLDA or SQLDA2 for which to deallocate data and indicator variables.

Errors

SQLSRV_ACTSTM	The statement id already has an active cursor.
SQLSRV_INTERR	Internal error.
SQLSRV_INVASC	Invalid association identifier.
SQLSRV_INVSQLDA	Invalid SQLDA, SQLDA2, or SQLDA_ID.
SQLSRV_MULTI_ACT	A batched sqlsrv_execute_in_out or sqlsrv_fetch_many context is active.
SQLSRV_SQLDA_NOTALL	Attempt to deallocate static memory.
SQLSRV_USRDATALL	The user, not Oracle SQL/Services, has allocated data buffers.

sqlsrv_set_option

The sqlsrv_set_option routine sets the option that determines whether the Oracle SQL/Services client and server use the standard SQLDA or the extended SQLDA2 format for new statements that the application prepares.

C Format

```
extern int sqlsrv_set_option(
                ASSOCIATE_ID association,
                SQS_LONGWORD option,
                SQS_LONGWORD value,
                void *rsv);
```

Parameters

association

An identifier used to distinguish one association from all others.

option

The option to set. The option parameter takes the argument SQLSRV_OPT_SQLDA_TYPE.

value

The value determines whether the SQLDA or SQLDA2 is set.

The value parameter takes either of the arguments described in Table 6–10 when the option parameter argument SQLSRV_OPT_SQLDA_TYPE is specified.

Table 6–10 Value Parameter Arguments If the Option Parameter Argument Is SQLSRV_OPT_SQLDA_TYPE

Argument	Description
SQLSRV_OPT_SQLDA_SQLDA	Use standard SQLDA format
SQLSRV_OPT_SQLDA_SQLDA2	Use extended SQLDA2 format

rsv

Argument reserved for future use. The value of this argument must be NULL.

Notes

If you do not call the sqlsrv_set_option routine to set the SQLDA format, Oracle SQL/Services uses the standard SQLDA format. To use the extended SQLDA2 format, you must call the sqlsrv_set_option routine, specifying the option as SQLSRV_OPT_SQLDA_TYPE and the value as SQLSRV_OPT_SQLDA_SQLDA2, before you call sqlsrv_prepare to prepare an SQL statement.

Errors

SQLSRV_INVARG	Invalid routine parameter.
SQLSRV_INVASC	Invalid association identifier.

6.3.5 Functional Interface Routines

Functional interface routines provide access to data and metadata stored in SQLCA, SQLDA, and SQLDA2 structures. These routines replace the need for making direct references to structure fields in API applications. Functional interface routines include the following routines:

- sqlsrv_sqlca_error routine (see sqlsrv_sqlca_error)
- sqlsrv_sqlca_error_text routine (see sqlsrv_sqlca_error_text)
- sqlsrv_sqlca_count routine (see sqlsrv_sqlca_count)
- sqlsrv_sqlca_sqlerrd routine (see sqlsrv_sqlca_sqlerrd)
- sqlsrv_sqlda_sqld or sqlsrv_sqlda2_sqld routine (see sqlsrv_sqlda_sqld or sqlsrv_sqlda2_sqld)
- sqlsrv_sqlda_column_name or sqlsrv_sqlda2_column_name routine (see sqlsrv_sqlda_column_name or sqlsrv_sqlda2_column_name)
- sqlsrv_sqlda_column_type or sqlsrv_sqlda2_column_type routine (see sqlsrv_sqlda_column_type or sqlsrv_sqlda2_column_type)
- sqlsrv_sqlda_bind_data or sqlsrv_sqlda2_bind_data routine (see sqlsrv_sqlda_bind_data or sqlsrv_sqlda2_bind_data)
- sqlsrv_sqlda_unbind_sqlda or sqlsrv_sqlda2_unbind_sqlda2 routine (see sqlsrv_sqlda_unbind_sqlda or sqlsrv_sqlda2_unbind_sqlda2)
- sqlsrv_sqlda_ref_data or sqlsrv_sqlda2_ref_data routine (see sqlsrv_sqlda_ref_data or sqlsrv_sqlda2_ref_data)
- sqlsrv_sqlda_unref_data or sqlsrv_sqlda2_unref_data routine (see sqlsrv_sqlda_unref_data or sqlsrv_sqlda2_unref_data)
- sqlsrv_sqlda_get_data or sqlsrv_sqlda2_get_data routine (see sqlsrv_sqlda_get_data or sqlsrv_sqlda2_get_data)
- sqlsrv_sqlda_set_data or sqlsrv_sqlda2_set_data routine (see sqlsrv_sqlda_set_data or sqlsrv_sqlda2_set_data)
- sqlsrv_sqlda_set_sqllen or sqlsrv_sqlda2_set_sqllen routine (see sqlsrv_sqlda_set_sqllen or sqlsrv_sqlda2_set_sqllen)
- sqlsrv_sqlda2_char_set_info routine (see sqlsrv_sqlda2_char_set_info)

sqlsrv_sqlca_error

The sqlsrv_sqlca_error routine returns the error codes for the last statement executed.

C Format

```
extern int sqlsrv_sqlca_error(
                ASSOCIATE_ID associate_id,
                SQS_LONGWORD *majerr,
                SQS_LONGWORD *suberr1,
                SQS_LONGWORD *suberr2);
```

Parameters

associate_id

An identifier used to distinguish one active association from all others.

majerr

Address of a variable of type SQS_LONGWORD into which the API writes the major error code from the SQLCODE field of the SQLCA.

suberr1

Address of a variable of type SQS_LONGWORD into which the API writes the minor error code from the SQLERRD[0] field of the SQLCA.

suberr2

Address of a variable of type SQS_LONGWORD into which the API writes the minor error code from the SQLERRD[2] field of the SQLCA.

Notes

After you call the Oracle SQL/Services API routine, the SQLCA structure contains the return status.

Errors

SQLSRV_INVASC	Invalid association identifier.

sqlsrv_sqlca_error_text

The sqlsrv_sqlca_error_text routine returns the error text for the last statement executed.

C Format

```
extern int sqlsrv_sqlca_error_text(
               ASSOCIATE_ID associate_id,
               short int *msglen,
               char *msg,
               short int buflen);
```

Parameters

associate_id

An identifier used to distinguish one active association from all others.

msglen

Address of a variable of type short into which the API writes the length in bytes of the error message text written to the buffer specified by the msg parameter.

msg

Address of a buffer of type char into which the API writes the error message text.

buflen

Length in bytes of the buffer specified by the msg parameter.

Notes

- The error message text is copied into the specified buffers and null-terminated.
- The length of the error excluding the null-terminator is returned in msglen.

sqlsrv_sqlca_count

The sqlsrv_sqlca_count routine returns the number of rows processed by a statement.

C Format

```
extern int sqlsrv_sqlca_count(
                ASSOCIATE_ID associate_id);
```

Parameters

associate_id

An identifier used to distinguish one active association from all others.

Notes

- This call replaces direct access to the SQLCA.SQLERRD[2] field.
- The SQLCA.SQLERRD[2] field contains a valid row count only when a statement, or all statements in a batch execute operation, executes successfully.

Errors

SQLSRV_INVASC	Invalid association identifier.

sqlsrv_sqlca_sqlerrd

The sqlsrv_sqlca_sqlerrd routine returns all values from the SQLCA.SQLERRD array.

C Format

```
extern int sqlsrv_sqlca_sqlerrd(
                ASSOCIATE_ID associate_id,
                SQS_LONGWORD *sqlerrd_array);
```

Parameters

associate_id

An identifier used to distinguish one active association from all others.

sqlerrd_array

Address of an array of 6 elements of type SQS_LONGWORD into which the API writes the contents of the SQLERRD array.

Notes

See Section 7.4 for details of information returned in the SQLERRD array.

Errors

SQLSRV_INVASC	Invalid association identifier.

sqlsrv_sqlda_sqld or sqlsrv_sqlda2_sqld

The sqlsrv_sqlda_sqld or sqlsrv_sqlda2_sqld routine returns the number of parameter markers or select list items in the SQLDA or SQLDA2.

Note

The format, parameters, description, notes, and errors for the SQLDA or SQLDA2 routines are identical unless otherwise specified.

C Format

```
extern int sqlsrv_sqlda_sqld(
                SQLDA_ID sqldaid);

extern int sqlsrv_sqlda2_sqld(
                SQLDA_ID sqldaid);
```

Parameters

sqldaid

The identifier of a parameter marker or select list SQLDA or SQLDA2.

Notes

This call corresponds to referencing the SQLD field in an SQLDA or SQLDA2. The field is set by the API after a statement is prepared.

Errors

SQLSRV_INVSQLDA	Invalid SQLDA, SQLDA2, or SQLDA_ID.

sqlsrv_sqlda_column_name or sqlsrv_sqlda2_column_name

The sqlsrv_sqlda_column_name or sqlsrv_sqlda2_column_name routine copies the column name for a particular column from the SQLDA or SQLDA2, respectively, into a program variable.

Note

The format, parameters, description, notes, and errors for the SQLDA or SQLDA2 routines are identical unless otherwise specified.

C Format

```
extern int sqlsrv_sqlda_column_name(
                SQLDA_ID sqldaid,
                short int colnum,
                char *colnam,
                short int *colnamlen);

extern int sqlsrv_sqlda2_column_name(
                SQLDA_ID sqldaid,
                short int colnum,
                char *colnam,
                short int *colnamlen);
```

Parameters

sqldaid

The identifier of a parameter marker or select list SQLDA or SQLDA2.

colnum

A column identified by its ordinal position in a parameter or select list.

colnam

Address of a buffer of type char into which the API writes the column name as a null-terminated character string. For an SQLDA, the buffer must be at least 30 bytes long; for an SQLDA2, the buffer must be at least 32 bytes long.

colnamlen

Address of a variable of type short into which the API writes the length in bytes of the column name written to the colnam parameter.

Notes

- Oracle SQL/Services returns an error if the SQLDA or SQLDA2 is invalid or if the column number is greater than the number of parameter markers or select list items (colnum >= sqlda.SQLD).
- The column name for a particular column is copied from the SQLDA into the variable passed in this call.
- Oracle Rdb does not assign a value to the column name in the following situations:
  - If a select list item, assignment, or comparison involves an arithmetic expression or predicates other than basic predicates.
  - For parameter markers and select list items specified in statements contained in a compound statement.
- The maximum length of a column name in an Oracle Rdb database is 31 characters. However, the maximum length of a column name stored by Oracle SQL/Services in the SQLNAME field of a client SQLDA is 29 characters. This is because the SQLNAME field is only 30 characters long and because Oracle SQL/Services null-terminates the column name in the SQLNAME field of a client SQLDA. The maximum length of a column name in the SQLNAME field of an Oracle SQL/Services client SQLDA2 is 31 characters.

Errors

SQLSRV_INVCOLNUM	Column number not within range.
SQLSRV_INVSQLDA	Invalid SQLDA, SQLDA2, or SQLDA_ID.

sqlsrv_sqlda_column_type or sqlsrv_sqlda2_column_type

The sqlsrv_sqlda_column_type or sqlsrv_sqlda2_column_type routine returns information about the data type of a column.

Note

The format, parameters, description, notes, and errors for the SQLDA and SQLDA2 routines are identical unless otherwise specified.

C Format

```
extern int sqlsrv_sqlda_column_type(
                SQLDA_ID sqldaid,
                short int colnum,
                short int *coltyp,
                unsigned short int *collen,
                short int *colscl,
                void *rsv);

extern int sqlsrv_sqlda2_column_type(
                SQLDA_ID sqldaid,
                short int colnum,
                short int *coltyp,
                SQS_UNSIGNED_LONGWORD *collen,
                short int *colscl,
                SQS_UNSIGNED_LONGWORD *coloctlen,
                void *rsv);
```

Parameters

sqldaid

The identifier of a parameter marker or select list SQLDA or SQLDA2.

colnum

A column identified by its ordinal position in a parameter or select list.

coltyp

Address of a variable of type short into which the API writes the Oracle SQL/Services data type of the column.

collen

Address of a variable into which the API writes the length of the column. For an SQLDA, the column length is expressed in an unsigned word as the number of 8-bit bytes. For an SQLDA2, the column length is expressed in an unsigned longword as the number of characters, where a single character might occupy more than one byte in a multibyte character set.

colscl

Address of a variable of type short into which the API writes the scale factor for columns of type SQLSRV_GENERALIZED_NUMBER or the type of date or interval for columns of type SQLSRV_GENERALIZED_DATE or SQLSRV_INTERVAL, respectively. Undefined for columns of all other data types.

coloctlen (SQLDA2 only)

Address of a variable of type SQS_UNSIGNED_LONGWORD into which the API writes the length of the column in octets or 8-bit bytes.

rsv

Argument reserved for future use. The value of this argument must be NULL.

Notes

- Oracle SQL/Services returns an error if the SQLDA or SQLDA2 is invalid or if the column number is greater than the number of parameter markers or select list items (colnum >= sqlda.SQLD).
- See Chapter 8 for information on all Oracle SQL/Services data types.

Errors

SQLSRV_INVCOLNUM	Column number not within range.
SQLSRV_INVSQLDA	Invalid SQLDA, SQLDA2, or SQLDA_ID.

sqlsrv_sqlda_bind_data or sqlsrv_sqlda2_bind_data

The sqlsrv_sqlda_bind_data or sqlsrv_sqlda2_bind_data routine allows programs to allocate their own storage for data and indicator variables for parameter markers and select list items.

Note

The format, parameters, description, notes, and errors for the SQLDA and SQLDA2 routines are identical unless otherwise specified.

C Format

```
extern int sqlsrv_sqlda_bind_data(
               SQLDA_ID sqldaid,
               short int colnum,
               short int coltyp,
               unsigned short int collen,
               short int colscl,
               CHARPTR datptr,
               SHORTPTR nulptr,
               void *rsv);

extern int sqlsrv_sqlda2_bind_data(
               SQLDA_ID sqldaid,
               short int colnum,
               short int coltyp,
               SQS_UNSIGNED_LONGWORD collen,
               short int colscl,
               CHARPTR datptr,
               LONGPTR nulptr,
               SQS_UNSIGNED_LONGWORD octet_len,
               SQS_LONGWORD chrono_scale,
               SQS_LONGWORD chrono_precision,
               void *rsv);
```

Parameters

sqldaid

The identifier of a parameter marker or select list SQLDA or SQLDA2.

colnum

A column identified by its ordinal position in a parameter or select list.

coltyp

Address of a variable of type short into which the API writes the Oracle SQL/Services data type of the column.

collen

Address of a variable into which the API writes the length of the column. For an SQLDA, the column length is expressed in an unsigned word as the number of 8-bit bytes. For an SQLDA2, the column length is expressed in an unsigned longword as the number of characters, where a single character might occupy more than one byte in a multibyte character set.

colscl

Address of a variable of type short into which the API writes the scale factor for columns of type SQLSRV_GENERALIZED_NUMBER or the type of date or interval for columns of type SQLSRV_GENERALIZED_DATE or SQLSRV_INTERVAL, respectively. This parameter is undefined for columns of all other data types.

datptr

Address of the data variable of type unsigned char for the column.

nulptr

Address of the indicator variable for the column. For an SQLDA, the indicator variable is of type short. For an SQLDA2, the indicator variable is of type SQS_LONGWORD. See Section 7.6 or Section 7.7 for a description of the indicator variable (SQLIND field) of an SQLDA or SQLDA2, respectively.

octet_len (SQLDA2 only)

Address of a variable of type SQS_UNSIGNED_LONGWORD into which the API writes the length in octets of the column.

chrono_scale (SQLDA2 only)

Address of a variable of type SQS_LONGWORD into which the API writes the specific date-time data type for columns of type SQLSRV_GENERALIZED_DATE or the interval scale for columns of type SQLSRV_INTERVAL.

chrono_precision (SQLDA2 only)

Address of a variable of type SQS_LONGWORD into which the API writes the precision of the date-time value or interval value for columns of type SQLSRV_GENERALIZED_DATE or SQLSRV_INTERVAL, respectively.

rsv

Argument reserved for future use. The value of this argument must be NULL.

Notes

- Oracle SQL/Services returns an error if the SQLDA or SQLDA2 is invalid or if the column number is greater than the number of parameter markers or select list items (colnum >= sqlda.SQLD).
- The sqlsrv_sqlda_bind_data and sqlsrv_sqlda2_bind_data routines provide an efficient mechanism for an application program to provide its own memory for data and indicator variables for parameter markers and select list items. After preparing a statement, the application must examine each column, allocate an appropriate amount of memory for both the data and indicator variables, then bind that memory to the column in the SQLDA or SQLDA2 using the sqlsrv_sqlda_bind_data or sqlsrv_sqlda2_bind_data routine, respectively. Before releasing the statement, the application program must unbind the memory for the column's data and indicator variables from the SQLDA or SQLDA2 using the sqlsrv_sqlda_unbind_data or sqlsrv_sqlda2_unbind_data routine, respectively.
- Applications that use the sqlsrv_sqlda_bind_data and sqlsrv_sqlda2_bind_data routines to provide memory for data and indicator variables in an SQLDA or SQLDA2 must allocate memory for all the parameter markers and select list items in the SQLDA or SQLDA. You cannot use the sqlsrv_allocate_sqlda_data or sqlsrv_allocate_sqlda2_data routines to allocate memory for the same SQLDA or SQLDA2 for which you have bound user memory to data and indicator variables.
- Calling the sqlsrv_sqlda_bind_data and sqlsrv_sqlda2_bind_data routines is equivalent to directly storing pointers and values in the SQLDATA, SQLIND, SQLLEN, and SQLOCTET_LEN fields of a column's SQLVARARY array element in an SQLDA or SQLDA2.

Errors

SQLSRV_INCDATTYP	Incompatible data type with column.
SQLSRV_INVCOLNUM	Column number not within range.
SQLSRV_INVDATTYP	Invalid data type.
SQLSRV_INVSQLDA	Invalid SQLDA, SQLDA2, or SQLDA_ID.
SQLSRV_NO_MEM	API memory allocation failed.

sqlsrv_sqlda_unbind_sqlda or sqlsrv_sqlda2_unbind_sqlda2

The sqlsrv_sqlda_unbind_sqlda or sqlsrv_sqlda2_unbind_sqlda2 routine releases variables bound with the sqlsrv_sqlda_bind_data or sqlsrv_sqlda2_bind_data routine.

Note

The format, parameters, description, notes, and errors for the SQLDA or SQLDA2 routines are identical unless otherwise specified.

C Format

```
extern int sqlsrv_sqlda_unbind_sqlda(
                SQLDA_ID sqldaid);

extern int sqlsrv_sqlda2_unbind_sqlda2(
                SQLDA_ID sqldaid);
```

Parameters

sqldaid

The identifier of a parameter marker or select list SQLDA or SQLDA2.

Notes

- A single call to sqlsrv_sqlda_unbind_data or sqlsrv_sqlda2_unbind_data unbinds the memory provided for all the data and indicator variables in an SQLDA or SQLDA2 bound by one or more calls to sqlsrv_sqlda_bind_data or sqlsrv_sqlda2_bind_data. On the Macintosh platform, it also releases memory allocated to maintain the application's memory pointers.
- Calling the sqlsrv_sqlda_bind_data and sqlsrv_sqlda2_bind_data routines is equivalent to directly clearing the pointers in the SQLDATA and SQLIND fields of a column's SQLVARARY array element in an SQLDA or SQLDA2.

Errors

SQLSRV_INVSQLDA	Invalid SQLDA, SQLDA2, or SQLDA_ID.

sqlsrv_sqlda_ref_data or sqlsrv_sqlda2_ref_data

The sqlsrv_sqlda_ref_data or sqlsrv_sqlda2_ref_data routine returns the type, length, scale, or date-time type, and address of the data and indicator variables for a column in an SQLDA or SQLDA2, respectively. In the SQLDA2, the sqlsrv_sqlda2_ref_data routine also returns the octet length, chrono-scale, and chrono-precision for a column.

Note

The format, parameters, description, notes, and errors for the SQLDA and SQLDA2 routines are identical unless otherwise specified.

C Format

```
extern int sqlsrv_sqlda_ref_data(
                SQLDA_ID sqldaid,
                short int colnum,
                short int *coltyp,
                unsigned short int *collen,
                short int *colscl,
                PTRCHARPTR val,
                PTRSHORTPTR nullp,
                void *rsv);

extern int sqlsrv_sqlda2_ref_data(
                SQLDA_ID sqldaid,
                short int colnum,
                short int *coltyp,
                SQS_UNSIGNED_LONGWORD *collen,
                short int *colscl,
                PTRCHARPTR val,
                PTRLONGPTR nullp,
                SQS_UNSIGNED_LONGWORD *octet_len,
                SQS_LONGWORD *chrono_scale,
                SQS_LONGWORD *chrono_precision,
                void *rsv);
```

Parameters

sqldaid

The identifier of a parameter marker or select list SQLDA or SQLDA2.

colnum

A column identified by its ordinal position in a parameter or select list.

coltyp

Address of a variable of type short into which the API writes the Oracle SQL/Services data type of the column.

collen

Address of a variable into which the API writes the length of the column. For an SQLDA, the column length is expressed in an unsigned word as the number of 8-bit bytes. For an SQLDA2, the column length is expressed in an unsigned longword as the number of characters, where a single character might occupy more than one byte in a multibyte character set.

colscl

Address of a variable of type short into which the API writes the scale factor for columns of type SQLSRV_GENERALIZED_NUMBER or the type of date or interval for columns of type SQLSRV_GENERALIZED_DATE or SQLSRV_INTERVAL, respectively. Undefined for columns of all other data types.

val

The address of a variable of type CHARPTR into which the API writes the address of the column's data variable.

nullp

Address of a variable into which the API writes the address of the column's indicator variable. For an SQLDA, the indicator variable is of type short. For an SQLDA2, the indicator variable is of type SQS_LONGWORD. See Section 7.6 or Section 7.7 for a description of the indicator variable (SQLIND field) of an SQLDA or SQLDA2, respectively.

octet_len (SQLDA2 only)

Address of a variable of type SQS_UNSIGNED_LONGWORD into which the API writes the length in octets of the column.

chrono_scale (SQLDA2 only)

Address of a variable of type SQS_LONGWORD into which the API writes the specific date-time data type for columns of type SQLSRV_GENERALIZED_DATE or the interval scale for columns of type SQLSRV_INTERVAL.

chrono_precision (SQLDA2 only)

Address of a variable of type SQS_LONGWORD into which the API writes the precision of the date-time value or interval value for columns of type SQLSRV_GENERALIZED_DATE or SQLSRV_INTERVAL, respectively.

rsv

Argument reserved for future use. The value of this argument must be NULL.

Notes

- Oracle SQL/Services returns an error if the SQLDA or SQLDA2 is invalid or if the column number is greater than the number of parameter markers or select list items (colnum >= sqlda.SQLD).
- Use the sqlsrv_sqlda_ref_data or sqlsrv_sqlda2_ref_data routine to access a column's data and indicator variables allocated by the sqlsrv_allocate_sqlda_data or sqlsrv_allocate_sqlda2_data routine. It is equivalent to reading the SQLLEN, SQLTYPE, SQLDATA, and SQLIND fields of the SQLVAR or SQLVAR2 structure, and for SQLDA2, the SQLOCTET_LEN, SQLCHRONO_SCALE, and SQLCHRONO_PRECISION fields of the SQLVAR2 structure for the column.
- Oracle SQL/Services transparently allocates memory for data and indicator variables using handles on the Macintosh platform. To provide a consistent, portable interface on the Macintosh platform with other client platforms, the sqlsrv_sqlda_ref_data and sqlsrv_sqlda2_ref_data routines lock the memory handles, then return the pointers, rather than the handles, to the data and indicator variables. Therefore, when your application has finished accessing data and indicator variables, you must call the sqlsrv_sqlda_unref_data and sqlsrv_sqlda2_unref_data routines to unlock the memory handles.

Errors

SQLSRV_INVCOLNUM	Column number not within range.
SQLSRV_INVSQLDA	Invalid SQLDA, SQLDA2, or SQLDA_ID.

sqlsrv_sqlda_unref_data or sqlsrv_sqlda2_unref_data

The sqlsrv_sqlda_unref_data or sqlsrv_sqlda2_unref_data routine frees resources tied up by the sqlsrv_sqlda_ref_data or sqlsrv_sqlda2_ref_data routine.

Note

The format, parameters, description, notes, and errors for the SQLDA or SQLDA2 routines are identical unless otherwise specified.

C Format

```
extern int sqlsrv_sqlda_unref_data(
                SQLDA_ID sqldaid,
                short int colnum);

extern int sqlsrv_sqlda2_unref_data(
                SQLDA_ID sqldaid,
                short int colnum);
```

Parameters

sqldaid

The identifier of a parameter marker or select list SQLDA or SQLDA2.

colnum

A column identified by its ordinal position in a parameter or select list.

Notes

- Oracle SQL/Services returns an error if the SQLDA or SQLDA2 is invalid or if the column number is greater than the number of parameter markers or select list items (colnum >= sqlda.SQLD).
- Oracle SQL/Services transparently allocates memory for data and indicator variables using handles on the Macintosh platform. To provide a consistent, portable interface on the Macintosh platform with other client platforms, the sqlsrv_sqlda_ref_data and sqlsrv_sqlda2_ref_data routines lock the memory handles, then return the pointers, rather than the handles, to the data and indicator variables. Therefore, when your application has finished

accessing data and indicator variables, you must call the sqlsrv_sqlda_unref_data and sqlsrv_sqlda2_unref_data routines to unlock the memory handles. On all other client platforms, the sqlsrv_sqlda_unref_data and sqlsrv_sqlda2_unref_data routines have no effect.

Errors

SQLSRV_INVCOLNUM	Column number not within range.
SQLSRV_INVSQLDA	Invalid SQLDA, SQLDA2, or SQLDA_ID.

sqlsrv_sqlda_get_data or sqlsrv_sqlda2_get_data

The sqlsrv_sqlda_get_data or sqlsrv_sqlda2_get_data routine copies column data and indicator variables from the SQLDA or SQLDA2, respectively, to a program.

Note

The format, parameters, description, notes, and errors for the SQLDA and SQLDA2 routines are identical unless otherwise specified.

C Format

```
extern int sqlsrv_sqlda_get_data(
                SQLDA_ID sqldaid,
                short int colnum,
                unsigned short int offset,
                CHARPTR dst,
                unsigned short int dstlen,
                SHORTPTR nullp,
                unsigned short int *bytcpy);

extern int sqlsrv_sqlda2_get_data(
                SQLDA_ID sqldaid,
                short int colnum,
                SQS_UNSIGNED_LONGWORD offset,
                CHARPTR dst,
                SQS_UNSIGNED_LONGWORD dstlen,
                LONGPTR nullp,
                SQS_UNSIGNED_LONGWORD *bytcpy);
```

Parameters

sqldaid

The identifier of a parameter marker or select list SQLDA or SQLDA2.

colnum

A column identified by its ordinal position in a parameter or select list.

offset

The offset within the column's data variable at which to start the copy. The most typical value for the offset parameter is zero, which means to start the copy at the beginning of the column's data variable. For an SQLDA, the offset is of type unsigned short. For an SQLDA2, the offset is of type SQS_UNSIGNED_LONGWORD.

dst

The address of a buffer of type unsigned char to which the data is copied.

dstlen

The length in bytes of the buffer specified as the dst argument. For an SQLDA, the length is of type unsigned short. For an SQLDA2, the length is of type SQS_UNSIGNED_LONGWORD.

nullp

Address of a variable into which Oracle SQL/Services writes the value of the column indicator variable. For an SQLDA, the indicator variable is of type short. For an SQLDA2, the indicator variable is of type SQS_LONGWORD. See Section 7.6 or Section 7.7 for a description of the indicator variable (SQLIND field) of an SQLDA or SQLDA2, respectively.

bytcpy

Address of a variable into which the API writes the number of bytes of data actually copied. For an SQLDA, the variable is of type unsigned short. For an SQLDA2, the variable is of type SQS_UNSIGNED_LONGWORD.

Notes

- Oracle SQL/Services returns an error if the SQLDA or SQLDA2 is invalid or if the column number is greater than the number of parameter markers or select list items (colnum >= sqlda.SQLD).
- The sqlsrv_sqlda_get_data or sqlsrv_sqlda2_get_data routine provides access to SQLDA or SQLDA2 information for languages that do not support explicit type coercion. Note that the use of the sqlsrv_sqlda_get_data or sqlsrv_sqlda2_get_data routine requires the host language to support some form of type coercion.
- When the sqlsrv_sqlda_get_data or sqlsrv_sqlda2_get_data routine is used, data is copied between the SQLDA or SQLDA2 and the user's buffer.

- The offset field provides some flexibility to callers, allowing you to take a selected section out of the field in question. The most typical value for the offset field is zero (0), which means to start copying at the beginning of the data. The maximum allowable value for the offset field is the maximum length of the SQLDATA buffer.

Errors

SQLSRV_INVCOLNUM	Column number not within range.
SQLSRV_INVSQLDA	Invalid SQLDA, SQLDA2, or SQLDA_ID.

sqlsrv_sqlda_set_data or sqlsrv_sqlda2_set_data

The sqlsrv_sqlda_set_data or sqlsrv_sqlda2_set_data routine copies column information into the SQLDA or SQLDA2, respectively.

Note

The format, parameters, description, notes, and errors for the SQLDA and SQLDA2 routines are identical unless otherwise specified.

C Format

```
extern int sqlsrv_sqlda_set_data(
                SQLDA_ID sqldaid,
                short int colnum,
                unsigned short int offset,
                CHARPTR dst,
                unsigned short int dstlen,
                short int nullp,
                unsigned short int *bytcpy);
extern int sqlsrv_sqlda2_set_data(
                SQLDA_ID sqldaid,
                short int colnum,
                SQS_UNSIGNED_LONGWORD offset,
                CHARPTR dst,
                SQS_UNSIGNED_LONGWORD dstlen,
                SQS_LONGWORD nullp,
                SQS_UNSIGNED_LONGWORD *bytcpy);
```

Parameters

sqldaid

The identifier of a parameter marker or select list SQLDA or SQLDA2.

colnum

A column identified by its ordinal position in a parameter or select list.

offset

The offset within the column's data variable at which to start the copy. The most typical value for the offset parameter is zero (0), which means to start the copy at the beginning of the column's data variable. For an SQLDA, the offset is of type unsigned short. For an SQLDA2, the offset is of type SQS_UNSIGNED_LONGWORD.

dst

The address of a buffer of type unsigned char containing the data to be copied to the SQLDATA buffer.

dstlen

The length in bytes of the buffer specified as the dst argument. For an SQLDA, the length is of type unsigned short. For an SQLDA2, the length is of type SQS_UNSIGNED_LONGWORD.

nullp

The value for the column's indicator variable. For an SQLDA, the indicator is of type short. For an SQLDA2, the indicator is of type SQS_LONGWORD. See Section 7.6 or Section 7.7 for a description of the indicator variable (SQLIND field) of an SQLDA or SQLDA2, respectively.

bytcpy

Address of a variable into which the API writes the number of bytes of data actually copied. For an SQLDA, the variable is of type unsigned short. For an SQLDA2, the variable is of type SQS_UNSIGNED_LONGWORD.

Notes

- Oracle SQL/Services returns an error if the SQLDA or SQLDA2 is invalid or if the column number is greater than the number of parameter markers or select list items (colnum >= sqlda.SQLD).
- The sqlsrv_sqlda_set_data or sqlsrv_sqlda2_set_data routine complements the sqlsrv_sqlda_get_data or sqlsrv_sqlda2_get_data routine. It is used to copy values into a column's data and indicator variables.
- The offset field provides some flexibility to callers, allowing you to target a selected section of the field in question. The most typical value for the offset field is zero (0), which means to target the copying at the beginning of the data. The maximum allowable value for the offset field is the maximum length of the SQLDATA or SQLIND buffer.

Errors

SQLSRV_INVCOLNUM	Column number not within range.
SQLSRV_INVSQLDA	Invalid SQLDA, SQLDA2, or SQLDA_ID.

sqlsrv_sqlda_set_sqllen or sqlsrv_sqlda2_set_sqllen

The sqlsrv_sqlda_set_sqllen or sqlsrv_sqlda2_set_sqllen routine sets the length of a column by setting the SQLLEN field in an SQLDA or the SQLLEN and SQLOCTET_LEN in an SQLDA2.

Note

The format, parameters, description, notes, and errors for the SQLDA and SQLDA2 routine are identical unless otherwise specified.

C Format

```
extern int sqlsrv_sqlda_set_sqllen(
                SQLDA_ID sqldaid,
                short int colnum,
                unsigned short int len);

extern int sqlsrv_sqlda2_set_sqllen(
                SQLDA_ID sqldaid,
                short int colnum,
                SQS_UNSIGNED_LONGWORD len,
                SQS_UNSIGNED_LONGWORD octet_len);
```

Parameters

sqldaid

The identifier of a parameter marker or select list SQLDA or SQLDA2.

colnum

A column identified by its ordinal position in a parameter or select list.

len

The length of the SQLLEN field in an SQLDA or SQLDA2.

octet_len (SQLDA2 only)

Address of a variable of type SQS_UNSIGNED_LONGWORD into which the API writes the length in octets of the column.

Notes

- Only columns of the SQLSRV_ASCII_STRING, SQLSRV_VARCHAR, and SQLSRV_VARBYTE data types can have their length changed.
- An SQLSRV_INVSETLEN error code is returned if you attempt to set the SQLLEN for a column of type SQLSRV_GENERALIZED_DATE, SQLSRV_GENERALIZED_NUMBER, SQLSRV_INTERVAL, or SQLSRV_LIST_VARBYTE.
- Use the sqlsrv_sqlda_set_sqllen or sqlsrv_sqlda2_set_sqllen routine to limit the amount of data returned in a column of a select list SQLDA. For example, if only the first few bytes of a column of type SQLSRV_ASCII_STRING, SQLSRV_VARCHAR, or SQLSRV_VARBYTE are required in certain circumstances, you can reduce the size of network messages by limiting the amount of data returned by the sqlsrv_fetch routine. When processing a call to sqlsrv_fetch or sqlsrv_execute_in_out, Oracle SQL/Services sends to the server only the lengths of those columns in a select list SQLDA or SQLDA2 that have changed since the last call.
- Use the sqlsrv_sqlda_set_sqllen or sqlsrv_sqlda2_set_sqllen routine to modify the length of a column of type SQLSRV_ASCII_STRING in a parameter marker SQLDA. In this situation, Oracle Rdb truncates or pads the value as necessary to the actual length of the column as specified in the database. Oracle SQL/Services does not need to send to the server the lengths of columns that have changed in a parameter marker SQLDA or SQLDA2, because the length of each data value is sent to the server along with the data itself.
- See Chapter 8 for more information on how Oracle SQL/Services handles values of each supported data type.
- You can increase or decrease the amount of memory Oracle SQL/Services allocates for a column by calling sqlsrv_sqlda_set_sqllen or sqlsrv_sqlda2_set_sqllen before you call sqlsrv_allocate_sqlda_data or sqlsrv_allocate_sqlda2_data. For example, Oracle Rdb allows you to store a segment of any length into a segmented string, regardless of the segment length specified in the database. Therefore, you may need to increase the length of a column of type SQLSRV_VARBYTE before you call sqlsrv_allocate_sqlda_data or sqlsrv_allocate_sqlda2_data to allocate the SQLDA data memory.
- For the sqlsrv_sqlda2_set_sqllen routine, the octlen parameter is compared with the len parameter to see if they are compatible. For example, the SQLLEN of a column of type SQLSRV_VARCHAR or SQLSRV_VARBYTE does not include the size of the leading 32-bit count field, whereas the SQLOCTET_LEN of a column of type SQLSRV_VARCHAR or SQLSRV_

VARBYTE does include the size of the leading 32-bit count field. If they are not compatible, an SQLSRV_INVSETLEN error code is returned.

When using a multibyte character set, normally the SQLLEN field represents the length in *characters* of a column, excluding the length of any control information, whereas the SQLOCTET_LEN represents the length in *bytes* of the column, including the length of any control information. However, Oracle SQL/Services does not send the SQLOCTET_LEN value to the server if it is changed; therefore, you must set the SQLLEN to the new length in *bytes* of the column, excluding the length of any control information.

Errors

SQLSRV_INVCOLNUM	Column number not within range.
SQLSRV_INVSQLDA	Invalid SQLDA, SQLDA2, or SQLDA_ID.
SQLSRV_INVDATTYP	Invalid data type.
SQLSRV_INVSETLEN	Unsupported data type or invalid SQLLEN and SQLOCTET_LEN combination.
SQLSRV_INVSQLLEN	The SQLLEN field in the SQLDA or SQLDA2 has been set to 0 or to a value greater than the size of the column.

sqlsrv_sqlda2_char_set_info

The sqlsrv_sqlda2_char_set_info routine returns the SQL character set fields from the SQLDA2.

C Format

```
extern int sqlsrv_sqlda2_char_set_info(
                SQLDA_ID sqldaid,
                short int colnum,
                CHARPTR name,
                short int name_len,
                CHARPTR schema,
                short int schema_len,
                CHARPTR catalog,
                short int catalog_len);
```

Parameters

sqldaid

The identifier of a parameter marker or select list SQLDA or SQLDA2.

colnum

A column identified by its ordinal position in a parameter or select list.

name

Address of a buffer of type unsigned char into which the API writes the character set name.

name_len

The length of the buffer specified by the name argument into which the API writes the character set name.

schema

Address of a buffer of type unsigned char into which the API writes the schema name.

schema_len

The length of the buffer specified by the schema argument into which the API writes the schema name.

catalog

Address of a buffer of type unsigned char into which the API writes the catalog name.

catalog_len

The length of the buffer specified by the catalog argument into which the API writes the catalog name.

Notes

- Oracle SQL/Services returns an error if the SQLDA or SQLDA2 is invalid or if the column number is greater than the number of parameter markers or select list items (colnum >= sqlda.SQLD).
- The maximum length of a character set name, schema name, or catalog name is 128 bytes. If a user-supplied buffer is smaller than the actual name, the name is truncated. If a user-supplied buffer is larger than the actual name, the name is padded with spaces.

Errors

SQLSRV_INVSQLDA	Invalid SQLDA, SQLDA2, or SQLDA_ID.
SQLSRV_INVCOLNUM	Column number not within range.

7
Data Structures

This chapter describes the data structures that Oracle SQL/Services uses to communicate with the client application. Some of the data structures (the SQLDA, SQLDA2, and SQLCA) are identical in layout (but not in usage) to those in dynamic SQL. Those structures are described in detail in the *Oracle Rdb7 SQL Reference Manual*. This Oracle SQL/Services manual provides relatively brief descriptions of the data structures and points out the differences in their usage.

7.1 Documentation Format

Each Oracle SQL/Services data structure is documented using a structured format called a template. The sections of the template are shown in Table 7–1, along with the information that is presented in each section and the format used to present the information.

Table 7–1 Sections in the Data Structure Template

Section	Description
Structure Name	Appears at the top of the page, followed by the English equivalent.
Overview	Appears directly below the structure name. The overview explains, usually in one or two sentences, the purpose of the structure.
Definition	Shows the C definition of the structure.
Fields	Gives detailed information about each field.

The Fields section contains detailed information about each field in the data structure. Fields are described in the order in which they appear in the structure.

Documentation Format

The following format is used to describe each field:

field-name

data type: The data type of the specific field (see Table 6–3)

C declaration: How that field is declared in the Oracle SQL/Services include files

set by: Whether the value of the field is set by the API, the application program, or both

used by: Whether the value of the field is used by the API, the application program, or both

In addition, the Fields section contains at least one paragraph of text describing the purpose of the field.

7.2 ASSOCIATE_STR—Association Structure

The association structure is a parameter that is passed to the sqlsrv_associate routine to specify the attributes of an association such as the service name, network transport, client logging flags, alternate error buffer, and so forth. ASSOCIATE_STR is defined in the include file sqlsrv.h.

```
struct ASSOCIATE_STR
    {
    unsigned short int  CLIENT_LOG;
    unsigned short      SERVER_LOG;
    short int           LOCAL_FLAG;
    short int           VERSION;
    CHARPTR             (*MEMORY_ROUTINE)();
    CHARPTR             (*FREE_MEMORY_ROUTINE)();
    short int           RESERVED;
    short int           ERRBUFLEN;
    CHARPTR             ERRBUF;
    CHARPTR             class_name;
    short int           xpttyp;
    short int           filler;
    CHARPTR             attach;
    CHARPTR             declare;
    CHARPTR             appnam;
    };
```

Fields

CLIENT_LOG

data type:	word (unsigned)
C declaration:	unsigned short int CLIENT_LOG
set by:	program
used by:	API

Specifies the type of client logging to be enabled on the client system (see Section 5.1).

The following constants are defined in the include file sqlsrv.h:

SQLSRV_LOG_DISABLED	Disables logging (default)
SQLSRV_LOG_ASSOCIATION	Enables association logging
SQLSRV_LOG_ROUTINE	Enables API routine logging
SQLSRV_LOG_PROTOCOL	Enables message protocol logging
SQLSRV_LOG_SCREEN[1]	Sends logging output to the video display on the client system as well as to the log file
SQLSRV_LOG_OPNCLS	Opens and closes the log file around each log file write and is useful if a client is terminated abnormally
SQLSRV_LOG_FLUSH	Flushes pending output to the log file only at the end of each complete association-level, routine-level, and protocol-level entry and is useful if a client application is terminating abnormally while executing application code.
SQLSRV_LOG_BINARY	Dumps memory in structured format if data contains non-printable characters

[1] See Chapter 5 for more information.

To enable more than one type of logging, add the appropriate constants.

SERVER_LOG

data type:	word (unsigned)
C declaration:	unsigned short int SERVER_LOG
set by:	program
used by:	unused

This feature is deprecated. This field is reserved.

LOCAL_FLAG

data type:	word (signed)
C declaration:	short int LOCAL_FLAG
set by:	program
used by:	unused

This feature is deprecated. This field is reserved.

VERSION

data type:	word (signed)
C declaration:	short int VERSION
set by:	program
used by:	API

Specifies the version of the ASSOCIATE_STR structure allocated by the application program. When set to a specific version number, such as SQLSRV_V700, the value of the VERSION field directs the client API to process fields in the ASSOCIATE_STR structure supported by the specified version. The SQLSRV_Vnnn version numbers are defined in sqlsrv.h.

MEMORY_ROUTINE

data type:	pointer
C declaration:	CHARPTR (*MEMORY_ROUTINE) ()
set by:	program
used by:	API

A pointer to the entry point of a user-specified routine to be called by the API for allocation of pointer-based memory. This feature is for client environments in which a limited amount of memory is available. The default value is NULL, which causes the API to use the portable C routine *malloc()* for pointer-based memory allocation. This value must be NULL for Macintosh systems, where the API uses the Macintosh NewHandle routine to allocate memory.

FREE_MEMORY_ROUTINE

data type:	pointer
C declaration:	CHARPTR (*FREE_MEMORY_ROUTINE) ()
set by:	program
used by:	API

A pointer to the entry point of a user-specified routine to be called by the API for deallocation of pointer-based memory. The default value is NULL, which causes the API to use the portable C routine *free()* for pointer-based memory deallocation. This value must be NULL for Macintosh systems, where the API uses the Macintosh DisposHandle routine to free memory.

RESERVED

data type:	word (signed)
C declaration:	short int RESERVED
set by:	program
used by:	unused

Must be 0. This field is reserved.

ERRBUFLEN

data type:	word (signed)
C declaration:	short int ERRBUFLEN
set by:	program
used by:	API

The length in bytes of an alternate error buffer specified by the ERRBUF field. Specify zero if you do not provide an alternate error buffer.

ERRBUF

data type:	pointer
C declaration:	CHARPTR ERRBUF
set by:	API
used by:	program

The address of an alternate error message buffer in which the API stores error message text. If you do not specify an alternate error message buffer, Oracle SQL/Services uses the 70-byte SQLERRMC field in the SQLCA data structure. However, because the SQLERRMC field is only 70 bytes, it may not be long enough to hold all the possible error messages that can be returned by the Oracle SQL/Services server or Oracle Rdb. Therefore, Oracle Corporation recommends that you allocate a larger message buffer for each association. A buffer of size 512 bytes is sufficient for all possible error messages.

class_name

data type:	pointer
C declaration:	CHARPTR class_name
set by:	program
used by:	API for VERSION SQLSRV_V610 and higher

The address of a buffer containing the service name with which to associate. This is the recommended method of choosing an Oracle SQL/Services service because it works for multiassociation applications. This field takes the place of the deprecated sqlsrv_set_server_class routine.

xpttyp

data type:	word (signed)
C declaration:	short int xpttyp
set by:	program
used by:	API for VERSION SQLSRV_V610 and higher

The desired transport type for this association. This is the recommended method of choosing a transport because it works for multiassociation applications. This field takes the place of the deprecated sqlsrv_set_transport_type routine.

The following constants are defined in the include file sqlsrv.h:

SQLSRV_XPT_NOT_CHOSEN	No transport chosen (default); API will select transport
SQLSRV_XPT_DECNET	Enables DECnet transport support
SQLSRV_XPT_TCPIP	Enables TCP/IP transport support
SQLSRV_XPT_ATK	Enables AppleTalk transport support (Macintosh only)
SQLSRV_XPT_SPXIPX	Enables IPX/SPX Novell NetWare transport support (MS Windows V3.1 only)
SQLSRV_XPT_SQLNET	Enables SQL*Net transport support

filler

data type:	word (signed)
C declaration:	short int filler
set by:	program
used by:	unused

Must be 0. This field is reserved.

attach

data type:	pointer
C declaration:	CHARPTR or an unsigned char*
set by:	program
used by:	API for VERSION SQLSRV_V610 and higher

Must be NULL, or set to a valid SQL ATTACH statement. You can use the attach field when associating to a universal service to avoid the extra round trip message to the server for an sqlsrv_execute_immediate call to issue the ATTACH statement. The ATTACH statement is executed in the executor after the SQL initialization procedure (if any) is executed.

declare

data type:	pointer
C declaration:	CHARPTR or an unsigned char*
set by:	program
used by:	API for VERSION SQLSRV_V610 and higher

Must be NULL, or any SQL statement that can be executed using sqlsrv_execute_immediate. The declare field is designed to specify a DECLARE TRANSACTION statement; however, you can specify any valid SQL statement. You can use the declare field when associating to a service of any type to avoid the extra round trip message to the server for an sqlsrv_execute_immediate call to issue a DECLARE TRANSACTION or other SQL statement. The SQL statement is executed in the executor after the SQL initialization procedure (if any) and ATTACH statement (if any) is executed.

appnam

data type:	pointer
C declaration:	CHARPTR or an unsigned char*
set by:	program
used by:	API for VERSION SQLSRV_V610 and higher

Must be NULL, or a string representing the client application name. Note that because the client application can pass any string using this field, the application name cannot be used for security purposes. The application name is displayed with a system management SHOW CLIENT command.

7.3 SQLCA—SQL Communications Area

The SQLCA structure is used to store information when an error occurs. This structure is defined in the include file sqlsrvca.h along with the error codes generated by Oracle SQL/Services.

```
struct SQLCA
    {
    char SQLCAID [8];
    SQS_LONGWORD SQLCABC;
    SQS_LONGWORD SQLCODE;
    struct
        {
        short int SQLERRML;
        char SQLERRMC [70];
        } SQLERRM;
    SQS_LONGWORD SQLERRD [6];
    struct
        {
        char SQLWARN0;
        char SQLWARN1;
        char SQLWARN2;
        char SQLWARN3;
        char SQLWARN4;
        char SQLWARN5;
        char SQLWARN6;
        char SQLWARN7;
        } SQLWARN;
    char SQLEXT [8];
    } ;
```

The Oracle SQL/Services SQLCA is based on the SQL SQLCA, which is described in detail in the *Oracle Rdb7 SQL Reference Manual*.

Fields

SQLCAID

data type:	character string
C declaration:	char SQLCAID [8]
set by:	API
used by:	unused

Structure identification field, present only for compatibility with SQL. Contains the null-terminated string “SQLCA” followed by two reserved bytes.

SQLCA—SQL Communications Area

SQLCABC

data type:	SQS_LONGWORD
C declaration:	SQS_LONGWORD SQLCABC
set by:	API
used by:	program

Contains the size, in bytes, of the SQLCA structure. The value of this field is always 128.

SQLCODE

data type:	SQS_LONGWORD
C declaration:	SQS_LONGWORD SQLCODE
set by:	API
used by:	program

Contains the error status for the most recently invoked Oracle SQL/Services routine. A positive value indicates a warning, a negative value indicates an error, and a 0 value indicates success. The include file sqlsrv.h contains the error messages that correspond to all of the possible values of SQLCODE returned by the Oracle SQL/Services client API.

SQLERRM.SQLERRML

data type:	word (signed)
C declaration:	short int SQLERRML
set by:	API
used by:	program

The length, in bytes, of the error message text returned in SQLERRMC.

SQLERRM.SQLERRMC

data type:	character string
C declaration:	char SQLERRMC [70]
set by:	API
used by:	program

The error message text, if any, that corresponds to the error contained in the SQLCODE field. This field is not used if you specify an alternate error message buffer. See Section 7.2 for more information.

SQLERRD

data type:	longword (signed) array
C declaration:	SQS_LONGWORD SQLERRD [6]
set by:	API
used by:	program

An array of six integers as described in Section 7.4.

SQLWARN*n*

data type:	character string
C declaration:	char SQLWARN0 . . . SQLWARN7
set by:	unused
used by:	unused

A series of eight 1-character state fields as defined by SQL.

SQLEXT

data type:	character string
C declaration:	char SQLEXT [8]
set by:	unused
used by:	unused

Not used by the API.

7.4 SQLERRD—Part of SQLCA

The SQLERRD array contains six elements. The content of each element in the SQLERRD array is determined by the routine that is successfully called:

- After a successful call to sqlsrv_prepare, the following information is stored in the SQLERRD array:

  SQLERRD[1] contains the statement type

  The statement types, as defined by Oracle Rdb, are as follows:

  0: statement is an executable statement other than CALL
  1: statement is a SELECT statement
  2: statement is a CALL statement

- After a successful call to sqlsrv_execute_immediate or sqlsrv_execute_in_out with the execute flag set to either SQLSRV_EXE_W_DATA or SQLSRV_EXE_WO_DATA, the following information is stored in the SQLERRD array:

  SQLERRD[2] element contains the number of rows inserted, updated, or deleted.

  See sqlsrv_execute_immediate and sqlsrv_execute_in_out for more information.

- After a successful call to sqlsrv_open_cursor to open a table cursor, the following information is stored in the SQLERRD array:

  SQLERRD[2] element contains the estimated result table cardinality.
  SQLERRD[3] element contains the estimated I/Os.

- After a successful call to sqlsrv_open_cursor to open a list cursor, the following information is stored to the SQLERRD array:

  SQLERRD[1] element contains the length of the largest actual segment.
  SQLERRD[3] element contains the total number of segments.
  The SQLERRD[4] and SQLERRD[5] elements contain the total length of all the segments as a quadword value where the low-order 32-bit value is stored in SQLERR[4] and the high-order 32-bit value is stored in SQLERRD[5].

- After a successful call to sqlsrv_fetch, the following information is stored in the SQLERRD array:

  SQLERRD[2] contains the number of the current row within the result table.

7.5 SQLDA or SQLDA2—SQL Descriptor Area

The SQLDA or SQLDA2 structure contains SQL parameter marker and select list metadata as well as pointers to data and indicator variables. It is defined in the include file sqlsrvda.h.

The Oracle SQL/Services SQLDA or SQLDA2 is identical to the SQLDA or SQLDA2 structures, respectively, in SQL. For additional information on the SQLDA or SQLDA2, read the dynamic SQL chapter in the *Oracle Rdb7 Guide to SQL Programming* and the SQLDA and SQLDA2 appendix in the *Oracle Rdb7 SQL Reference Manual*.

```
struct SQLDA
    {
    char            SQLDAID[8];
    SQS_LONGWORD    SQLDABC;
    unsigned short  SQLN;
    unsigned short  SQLD;
    struct SQLVAR SQLVARARY[1];
    };
```

```
struct SQLDA2
    {
    char            SQLDAID[8];
    SQS_LONGWORD    SQLDABC;
    unsigned short  SQLN;
    unsigned short  SQLD;
    struct SQLVAR2 SQLVARARY[1];
    };
```

Fields

SQLDAID

data type:	character string
C declaration:	char SQLDAID[8]
set by:	API
used by:	unused

Structure identification field; contains the null-terminated string “SQLDA” or “SQLDA2” followed by one or two reserved bytes.

SQLDABC

data type:	SQS_LONGWORD
C declaration:	SQS_LONGWORD SQLDABC
set by:	API or program
used by:	API

The size, in bytes, of the SQLDA or SQLDA2 structure, including the nested variable length SQLVARARY structure. The SQLDABC field is used by the API to verify the integrity of the SQLDA or SQLDA2.

SQLN

data type:	word (signed)
C declaration:	short int SQLN
set by:	see following text
used by:	API

The number of elements in the SQLVARARY. If the API allocated the SQLDA or SQLDA2 structure, this value is the same as the SQLD field. If your application allocated its own SQLDA or SQLDA2 structure, it must supply this value. In that case, the SQLN field specifies the maximum number of select list items or parameter marker items that can exist in an SQL statement that is prepared with a particular SQLDA or SQLDA2; a call to the sqlsrv_prepare routine with an SQLVARARY that is too small returns an error.

SQLD

data type:	word (signed)
C declaration:	short int SQLD
set by:	API
used by:	program

The actual number of parameter markers or select list items in a prepared SQL statement. In an SQLDA or SQLDA2 structure that was allocated by the API, this value is the same as the SQLN field (the number of elements in the SQLVARARY).

SQLVARARY

data type:	structure array
C declaration:	struct SQLVAR SQLVARARY[1] (SQLDA), struct SQLVAR2 SQLVARARY[1] (SQLDA2)
set by:	see Section 7.6 and Section 7.7
used by:	see Section 7.6 and Section 7.7

An array of SQLVAR structures (see Section 7.6) or SQLVAR2 structures (see Section 7.7), each of which describes one select list item or one parameter marker item. Because some C compilers do not support the definition of a varying array within a structure, SQLVARARY is defined as an array of one element. However, Oracle SQL/Services uses as many SQLVAR or SQLVAR2 elements as allocated in an SQLDA or SQLDA2.

7.6 SQLVAR—Parameter Marker or Select List Item

Each SQLVAR structure describes one select list item or parameter marker.

```
struct SQLVAR
    {
    short           SQLTYPE;
    unsigned short  SQLLEN;
    CHARPTR         SQLDATA;
    SHORTPTR        SQLIND;
    short           SQLNAME_LEN;
    char            SQLNAME[30];
    };
```

Fields

SQLTYPE

data type:	word (signed)
C declaration:	short int SQLTYPE
set by:	API
used by:	program

The SQL data type for the SQLVAR entry. This value represents the Oracle SQL/Services data type as defined in the include file sqlsrv.h.

```
#define SQLSRV_ASCII_STRING                    129
#define SQLSRV_GENERALIZED_NUMBER              130
#define SQLSRV_GENERALIZED_DATE                131
#define SQLSRV_VARCHAR                         132
#define SQLSRV_VARBYTE                         155
#define SQLSRV_LIST_VARBYTE                    159
#define SQLSRV_INTERVAL                        168
```

SQLLEN

data type:	word (signed)
C declaration:	unsigned short int SQLLEN
set by:	see following text
used by:	program

The value of the SQLLEN field is dependent on the data type of the parameter marker or select list item. For more information, see Chapter 8.

SQLDATA

data type:	pointer
C declaration:	char *SQLDATA
set by:	program or API

used by:	program and API

The address of the data variable for the parameter marker or select list item. If your application allocates data variables by calling the sqlsrv_allocate_sqlda_data or sqlsrv_allocate_sqlda2_data routine, the API initializes this field. If your application allocates its own data variables, it must write the address of each variable into an SQLDATA field. In that case, the API returns an error if an SQLLEN value is less than the length of the associated data value.

SQLIND

data type:	pointer
C declaration:	short int *SQLIND
set by:	program or API
used by:	program and API

The address of the indicator variable for the data. If your application calls the sqlsrv_allocate_sqlda_data or sqlsrv_allocate_sqlda2_data routine, the API initializes this field. Otherwise, your application must allocate its own indicator variables and write the address of each variable into an SQLIND field.

Your program sets the indicator variable of each parameter marker as follows before calling sqlsrv_execute_in_out or sqlsrv_open_cursor:

0: to indicate the presence of data for the column
–1: to indicate a NULL value for the column

The API sets the indicator variable of each select list item as follows as part of the successful completion of a call to sqlsrv_fetch or sqlsrv_execute_in_out:

0: to indicate the presence of data for the column
–1: to indicate a NULL value for the column
>0: to indicate that a column value was truncated

SQLNAME_LEN

data type:	word (signed)
C declaration:	short int SQLNAME_LEN
set by:	API
used by:	program

The length, in bytes, of the name stored in the SQLNAME field.

SQLVAR—Parameter Marker or Select List Item

SQLNAME

data type:	character string
C declaration:	char SQLNAME[30]
set by:	API
used by:	program

The name of the parameter marker or select list item. Oracle SQL/Services stores the name as a null-terminated string.

7.7 SQLVAR2—Parameter Marker or Select List Item

Each SQLVAR2 structure describes one select list item or parameter marker.

```
struct SQLVAR2
    {
    short                   SQLTYPE;
    SQS_UNSIGNED_LONGWORD   SQLLEN;
    SQS_UNSIGNED_LONGWORD   SQLOCTET_LEN;
    CHARPTR                 SQLDATA;
    LONGPTR                 SQLIND;
    SQS_LONGWORD            SQLCHRONO_SCALE;
    SQS_LONGWORD            SQLCHRONO_PRECISION;
    short                   SQLNAME_LEN;
    char                    SQLNAME[128];
    char                    SQLCHAR_SET_NAME[128];
    char                    SQLCHAR_SET_SCHEMA[128];
    char                    SQLCHAR_SET_CATALOG[128];
    };
```

Fields

SQLTYPE

data type:	word (signed)
C declaration:	short int SQLTYPE
set by:	API
used by:	program

The SQL data type for the SQLVAR2 entry. This value represents the Oracle SQL/Services data type as defined in the include file sqlsrv.h.

```
#define SQLSRV_ASCII_STRING                     129
#define SQLSRV_GENERALIZED_NUMBER               130
#define SQLSRV_GENERALIZED_DATE                 131
#define SQLSRV_VARCHAR                          132
#define SQLSRV_VARBYTE                          155
#define SQLSRV_LIST_VARBYTE                     159
#define SQLSRV_INTERVAL                         168
```

SQLLEN

data type:	SQS_LONGWORD_UNSIGNED
C declaration:	SQS_LONGWORD_UNSIGNED SQLLEN
set by:	see following text
used by:	program

The value of the SQLLEN field is dependent on the data type of the parameter marker or select list item. For more information, see Chapter 8.

SQLOCTET_LEN

data type:	SQS_LONGWORD_UNSIGNED
C declaration:	SQS_LONGWORD_UNSIGNED SQLOCTET_LEN
set by:	SQL
used by:	program and API

A value that indicates the length in octets or 8-bit bytes of the select list item or parameter marker. For more information, see Chapter 8.

SQLDATA

data type:	pointer
C declaration:	char *SQLDATA
set by:	program or API
used by:	program and API

The address of the data variable for the parameter marker or select list item. If your application allocates data variables by calling the sqlsrv_allocate_sqlda_data or sqlsrv_allocate_sqlda2_data routine, the API initializes this field. If your application allocates its own data variables, it must write the address of each variable into an SQLDATA field. In that case, the API returns an error if an SQLLEN value is less than the length of the associated data value.

SQLIND

data type:	pointer
C declaration:	SQS_LONGWORD *SQLIND
set by:	program or API
used by:	program and API

The address of the indicator variable for the data. If your application calls the sqlsrv_allocate_sqlda_data or sqlsrv_allocate_sqlda2_data routine, the API initializes this field. Otherwise, your application must allocate its own indicator variables and write the address of each variable into an SQLIND field.

Your program sets the indicator variable of each parameter marker as follows before calling sqlsrv_execute_in_out or sqlsrv_open_cursor:

0: to indicate the presence of data for the column
–1: to indicate a NULL value for the column

The API sets the indicator variable of each select list item as follows as part of the successful completion of a call to sqlsrv_fetch or sqlsrv_execute_in_out:

0: to indicate the presence of data for the column
–1: to indicate a NULL value for the column

>0: to indicate that a column value was truncated

SQLCHRONO_SCALE

data type:	SQS_LONGWORD
C declaration:	SQS_LONGWORD SQLCHRONO_SCALE
set by:	API
used by:	program

SQLCHRONO_SCALE contains the scale of the interval for columns of type SQLSRV_INTERVAL. SQLCHRONO_SCALE contains the type of date as shown in Table 8–2 for columns of type SQLSRV_GENERALIZED_DATE.

SQLCHRONO_PRECISION

data type:	SQS_LONGWORD
C declaration:	SQS_LONGWORD SQLCHRONO_PRECISION
set by:	API
used by:	program

SQLCHRONO_PRECISION contains the precision for columns of type SQLSRV_INTERVAL and for columns of type SQLSRV_GENERALIZED_DATE with a type of SQLSRV_DT_DATE_ANSI, SQLSRV_DT_TIME, or SQLSRV_DT_TIMESTAMP.

SQLNAME_LEN

data type:	word (signed)
C declaration:	short int SQLNAME_LEN
set by:	API
used by:	program

The length, in bytes, of the name stored in the SQLNAME field.

SQLNAME

data type:	character string
C declaration:	char SQLNAME[128]
set by:	API
used by:	program

The name of the parameter marker or select list item. Oracle SQL/Services stores the name as a null-terminated string. The maximum length of a name is 31 characters.

SQLCHAR_SET_NAME

data type:	character string
C declaration:	char SQLCHAR_SET_NAME[128]
set by:	API
used by:	program

The character set name when the SQLTYPE is a character string type. The maximum length of a character set name is 128 characters. When SQLTYPE is any other data type, this field contains spaces.

SQLCHAR_SET_SCHEMA

data type:	character string
C declaration:	char SQLCHAR_SET_SCHEMA[128]
set by:	reserved for future use
used by:	reserved for future use

The schema name when the SQLTYPE is a character string type. The maximum length of a schema name is 128 characters. When SQLTYPE is any other data type, this field contains spaces.

SQLCHAR_SET_CATALOG

data type:	character string
C declaration:	char SQLCHAR_SET_CATALOG[128]
set by:	reserved for future use
used by:	reserved for future use

The catalog name when the SQLTYPE is a character string type. The maximum length of a catalog name is 128 characters. When SQLTYPE is any other data type, this field contains spaces.

8
Data Types

Oracle SQL/Services supports the full range of SQL data types; however, the values for certain data types are represented in a different format than that used in the database. Each SQL data type has a corresponding Oracle SQL/Services data type, all of which are described in this chapter. The sqlsrv.h file provides definitions for each Oracle SQL/Services data type.

8.1 Data Types

Table 8–1 lists the SQL data types along with the corresponding Oracle SQL/Services data types.

Table 8–1 Data Types

SQL Data Type	Oracle SQL/Services Data Type
CHAR	SQLSRV_ASCII_STRING
VARCHAR	SQLSRV_VARCHAR
TINYINT	SQLSRV_GENERALIZED_NUMBER
SMALLINT	SQLSRV_GENERALIZED_NUMBER
INTEGER	SQLSRV_GENERALIZED_NUMBER
QUADWORD	SQLSRV_GENERALIZED_NUMBER
FLOAT	SQLSRV_GENERALIZED_NUMBER
REAL	SQLSRV_GENERALIZED_NUMBER
DOUBLE PRECISION	SQLSRV_GENERALIZED_NUMBER

(continued on next page)

Table 8–1 (Cont.) Data Types

SQL Data Type	Oracle SQL/Services Data Type
DATE VMS	SQLSRV_GENERALIZED_DATE
DATE ANSI	SQLSRV_GENERALIZED_DATE
TIME	SQLSRV_GENERALIZED_DATE
TIMESTAMP	SQLSRV_GENERALIZED_DATE
INTERVAL	SQLSRV_INTERVAL
LIST OF BYTE VARYING	SQLSRV_LIST_VARBYTE
String segment data type	SQLSRV_VARBYTE

8.2 SQLSRV_ASCII_STRING

Oracle SQL/Services uses the SQLSRV_ASCII_STRING data type to represent the CHAR fixed-length character string data type.

For an SQLDA, the SQLLEN field specifies the length of the string in 8-bit bytes. For an SQLDA2, the SQLLEN field specifies the length of the string in characters and the SQLOCTET_LEN field specifies the length of the string in 8-bit bytes.

If the client application calls either the sqlsrv_allocate_sqlda_data( ) or sqlsrv_allocate_sqlda2_data( ) client API service to allocate the SQLDATA memory, then Oracle SQL/Services allocates an extra byte of memory and null-terminates SQLSRV_ASCII_STRING character strings in select list SQLDAs. The extra byte of memory is not reflected in the SQLLEN or SQLOCTET_LEN fields. If the client application allocates its own SQLDATA memory, then Oracle SQL/Services does not null-terminate SQLSRV_ASCII_STRING character strings.

8.3 SQLSRV_VARCHAR

Oracle SQL/Services uses the SQLSRV_VARCHAR data type to represent the VARCHAR varying-length string data type. An SQLSRV_VARCHAR data value consists of a leading length field immediately followed by the string, which may contain binary data.

For an SQLDA, the leading length field is an unsigned 16-bit word. The SQLLEN field specifies the maximum length of a string in 8-bit bytes, excluding the size of the 16-bit leading length field.

For an SQLDA2, the leading length field is an unsigned 32-bit longword. The SQLLEN field specifies the maximum length of a string in characters, excluding the size of the 32-bit leading length field. The SQLOCTET_LEN field specifies the maximum length of a string in 8-bit bytes, including the size of the 32-bit leading length field.

8.4 SQLSRV_GENERALIZED_NUMBER

Oracle SQL/Services uses the SQLSRV_GENERALIZED_NUMBER data type to represent the following SQL data types:

- TINYINT
- SMALLINT
- INTEGER
- QUADWORD
- FLOAT
- REAL
- DOUBLE PRECISION

Oracle SQL/Services presents all integer, fixed-point, and floating-point data values to a client application as null-terminated numeric strings in the following format:

```
[-][NNN][.DD][E[-][xx]]
```

–	unary minus
NNN	integer portion of the number
.DD	decimal portion of the number
E	exponent identifier
–	unary minus for exponent value
xx	exponent value

The brackets indicate the optional syntax.

When you prepare a statement, the Oracle SQL/Services executor calculates the maximum number of bytes required to represent the most negative and the most positive value for an Oracle SQL/Services generalized number.

For an SQLDA, the low-order byte of the SQLLEN field specifies the maximum number of bytes, excluding the null-terminator. The high-order byte of the SQLLEN field specifies the scale factor.

For an SQLDA2, the low-order 16-bit word of the SQLLEN field specifies the maximum number of bytes, excluding the null-terminator. The high-order 16-bit word of the SQLLEN field specifies the scale factor. The SQLOCTET_LEN field specifies the maximum number of bytes, including the null-terminator.

SQL allows a parameter marker value for an integer or fixed-point data type to be supplied in scientific notation. For example: –3.2768E4 is equivalent to –32768. To support this, the sqlsrv_allocate_sqlda_data( ) and sqlsrv_allocate_sqlda2_data( ) client API services both allocate an additional 5 bytes of memory to account for a possible decimal point (.) and exponent (E+nn). These 5 extra bytes are not reflected in either the SQLLEN or SQLOCTET_LEN values.

It is possible for an application that allocates its own memory for parameter marker data variables to send a numeric data value to the server that is a valid number, but that is potentially longer than the server can handle. For this reason, the server allocates an extra 10 bytes of memory for parameter marker variables for all numeric data types, in addition to the minimum required for each data type. If the length of a numeric parameter marker value exceeds the amount of memory allocated for the parameter marker variable, the server returns the SQLSRV_DATA_TOO_LONG error to the client. This restriction is imposed on the server by the particular dynamic SQL interface used by the Oracle SQL/Services server.

For example, the server minimally allocates 6 bytes for a column of type SMALLINT. This supports values from –32768 through +32767. To handle values expressed in scientific notation, the server allocates an additional 5 bytes for all numeric data types. This supports values from -3.2768E+04 through +3.2767E+04. To support the inclusion of insignificant zeros, the server finally allocates an additional 10 bytes for all numeric data types. This supports values such as –003.276800E+4 and +3.2767E+0004. However, a value of +00003.27670000E+00004, although a valid numeric value, is considered too long to be handled by the server.

8.5 SQLSRV_GENERALIZED_DATE

Oracle SQL/Services uses the SQLSRV_GENERALIZED_DATE data type to represent the DATE VMS, DATE ANSI, TIME, and TIMESTAMP data types. An Oracle SQL/Services generalized date is a null-terminated string containing a maximum of 16 digits in the following format:

```
ccyymmdd[hh[mi[ss[ff]]]]
```

cc	century
yy	year

mm	month
dd	day
hh	hour (24-hour format)
mi	minute
ss	second
ff	fractions of a second

If you omit any of the optional fields of a date-time value of type DATE VMS, then SQL pads the string with zeros. Thus, the default time is exactly midnight.

In a select list SQLDA, the century, year, month, and day fields of a date-time value of type TIME are all zeros. In a parameter marker SQLDA, the century, year, month, and day fields of a date-time value of type TIME are ignored, but must be present. Oracle Corporation recommends you set these fields to all zeros.

In a select list SQLDA, the hours, minutes, seconds, and fractions-of-second fields of a date-time value of type DATE ANSI are all zeros. In a parameter marker SQLDA, the hours, minutes, seconds, and fractions-of-second fields of a date-time value of type DATE ANSI are ignored.

All the fields of date-time value of type TIMESTAMP are significant in both select list and parameter marker SQLDAs. For example:

Data Type	Date/Time	SQLSRV_GENERALIZED_DATE
DATE VMS	June 26, 1961 11:04:05 AM	1961062611040500
DATE ANSI	March 22, 1996	1996032200000000
TIME	11:23:06.7 AM	0000000011230670
TIMESTAMP	May 6, 1994 2:34:56.21 PM	1994050614345621

For an SQLDA, the low-order byte of the SQLLEN field specifies the maximum number of digits, including the null-terminator. Thus the value is always 17. The high-order byte of the SQLLEN field specifies the Oracle SQL/Services date-time data type as shown in Table 8–2. The precision of the fractions-of-second field of a date-time value of type TIME or TIMESTAMP value is not available for an SQLDA.

For an SQLDA2, the SQLLEN and SQLOCTET_LEN fields both contain the maximum number of digits, including the null-terminator. Thus both values are always 17. The SQLCHRONO_SCALE field specifies the Oracle SQL/Services date-time data type as shown in Table 8–2. The SQLCHRONO_

PRECISION field specifies the precision of the fractions-of-second field. This value is undefined for a date-time value of type DATE VMS.

Table 8–2 Oracle SQL/Services Date-Time Data Types

Value	Oracle SQL/Services Date-Time Data Types	SQL Date-Time Data Types
0	SQLSRV_DT_DATE_VMS	DATE VMS
1	SQLSRV_DT_DATE_ANSI	DATE ANSI
2	SQLSRV_DT_TIME	TIME
3	SQLSRV_DT_TIMESTAMP	TIMESTAMP

It is possible for an application that allocates its own memory for parameter marker data variables to send a date-time data value to the server that is valid, but that is potentially longer than the server can handle. For this reason, the server allocates an extra 10 bytes of memory for parameter marker variables for all date-time data types, in addition to the minimum required for each data type. If the length of a date-time parameter marker value exceeds the amount of memory allocated for the parameter marker variable, the server returns the SQLSRV_DATA_TOO_LONG error to the client. This restriction is imposed on the server by the particular dynamic SQL interface used by the Oracle SQL/Services server.

For example, the server minimally allocates 16 bytes for a column of type TIMESTAMP. This supports all valid timestamp values expressed in the Oracle SQL/Services generalized date format, such as 1996073009572249 (1996-07-30:09:57:22.49). To support the inclusion of insignificant zeros, the server also allocates an additional 10 bytes for all date-time data types. This supports values such as 199607300957224900. However, a value of 19960730095722490000000000, although a valid date-time value, is considered too long to be handled by the server.

See the *Oracle Rdb7 SQL Reference Manual* and the *Oracle Rdb7 Guide to SQL Programming* for more information on the SQL date-time data types.

8.6 SQLSRV_INTERVAL

Oracle SQL/Services uses the SQLSRV_INTERVAL data type to represent the INTERVAL data type. An Oracle SQL/Services interval is a null-terminated string.

When you prepare a statement, the Oracle SQL/Services executor calculates the maximum number of bytes required to represent the most negative and the most positive value for an interval.

For an SQLDA, the low-order byte of the SQLLEN field specifies the maximum number of bytes, excluding the null-terminator. The high-order byte of the SQLLEN field specifies the interval subtype. The scale and precision of the interval are not available for an SQLDA.

For an SQLDA2, the SQLLEN field specifies the interval subtype. The SQLOCTET_LEN field specifies the maximum number of bytes, including the null-terminator. The scale and precision of the interval are specified by the SQLCHRONO_SCALE and SQLCHRONO_PRECISION fields, respectively.

The Oracle SQL/Services interval codes shown in Table 8–3 correspond directly to the SQL interval types.

Table 8–3 Oracle SQL/Services Interval Type

Value	Oracle SQL/Services Interval Type
1	SQLSRV_DT_YEAR
2	SQLSRV_DT_MONTH
3	SQLSRV_DT_DAY
4	SQLSRV_DT_HOUR
5	SQLSRV_DT_MINUTE
6	SQLSRV_DT_SECOND
7	SQLSRV_DT_YEAR_MONTH
8	SQLSRV_DT_DAY_HOUR
9	SQLSRV_DT_DAY_MINUTE
10	SQLSRV_DT_DAY_SECOND
11	SQLSRV_DT_HOUR_MINUTE
12	SQLSRV_DT_HOUR_SECOND
13	SQLSRV_DT_MINUTE_SECOND

It is possible for an application that allocates its own memory for parameter marker data variables to send an interval data value to the server that is valid, but that is potentially longer than the server can handle. For this reason, the server allocates an extra 10 bytes of memory for parameter marker variables for all interval data types, in addition to the minimum required for each data type. If the length of an interval parameter marker value exceeds the amount of memory allocated for the parameter marker variable, the server returns the

SQLSRV_DATA_TOO_LONG error to the client. This restriction is imposed on the server by the particular dynamic SQL interface used by the Oracle SQL/Services server.

For example, the server minimally allocates 3 bytes for a column of type INTERVAL YEAR(3). This supports values from –99 through 99. To support the inclusion of insignificant zeros, the server also allocates an additional 10 bytes for all interval data types. This supports values such as +000999, although SQL may consider insignificant zeros as invalid. However, a value of –0000000000099, although potentially a valid interval value, is considered too long to be handled by the server.

See the *Oracle Rdb7 Guide to SQL Programming* and the *Oracle Rdb7 SQL Reference Manual* for more information on the INTERVAL data type.

8.7 SQLSRV_VARBYTE

Oracle SQL/Services uses the SQLSRV_VARBYTE data type to represent the varying-length string segment data type. An SQLSRV_VARBYTE data value consists of a leading length field immediately followed by the string, which may contain binary data.

For an SQLDA, the leading length field is an unsigned 16-bit word. The SQLLEN field specifies the maximum length of a string in 8-bit bytes, excluding the size of the 16-bit leading length field.

For an SQLDA2, the leading length field is an unsigned 32-bit longword. The SQLLEN field specifies the maximum length of a string in characters, excluding the size of the 32-bit leading length field. The SQLOCTET_LEN field specifies the maximum length of a string in 8-bit bytes, including the size of the 32-bit leading length field.

When dealing with the SQLSRV_VARBYTE data type, it is important to know that the length of a segment may exceed the length specified in the metadata for a column. For example, the default segment length is 1 byte; however, segments of any length may be stored in a column defined with the default length. Consider a segmented string defined as LIST OF BYTE VARYING(80).

In a parameter marker SQLDA, you can call sqlsrv_sqlda_set_sqllen( ) or sqlsrv_sqlda2_set_sqllen( ) to increase the maximum segment length to 132 bytes before you call sqlsrv_allocate_sqlda_data( ) or sqlsrv_allocate_sqlda2_data( ). You may then insert strings up to 132 bytes in length into the segmented string.

In a select list SQLDA, sqlsrv_prepare( ) returns the segment length specified when the column was defined. In this example, the column was defined with a segment length of 80 bytes. However, the length of the longest segment in a particular segmented string may be longer than this value. In this example, it is 132 bytes. To allow your application to allocate sufficient memory for the longest segment, sqlsrv_open_cursor( ) returns the length of the longest segment in the SQLERRD[1] field of the SQLCA when you successfully open a list cursor to access the segmented string. Therefore, in this example, sqlsrv_open_cursor( ) returns 132 in the SQLERRD[1] field. You can then supply this value to sqlsrv_sqlda_set_sqllen( ) or sqlsrv_sqlda2_set_sqllen( ) before you call sqlsrv_allocate_sqlda_data( ) or sqlsrv_allocate_sqlda2_data( ). In this way, you are guaranteed to have sufficient SQLDATA memory available to hold the longest segment in the segment string.

See the *Oracle Rdb7 Guide to SQL Programming* and the *Oracle Rdb7 SQL Reference Manual* for more information on lists (segmented strings).

8.8 SQLSRV_LIST_VARBYTE

Oracle SQL/Services uses the SQLSRV_LIST_VARBYTE data type to represent the LIST OF BYTE VARYING data type. The SQLSRV_LIST_VARBYTE data type is a fixed-length data type that holds the location of a particular segemented string or binary large object (BLOB) in a database.

For an SQLDA, the SQLLEN field specifies the size in bytes of the SQLSRV_LIST_VARBYTE.

For an SQLDA2, both the SQLLEN and SQLOCTET_LEN fields specify the size in bytes of the SQLSRV_LIST_VARBYTE.

See the *Oracle Rdb7 Guide to SQL Programming* and the *Oracle Rdb7 SQL Reference Manual* for more information on the LIST OF BYTE VARYING data type.

8.9 Deciding Whether to Use SQLDA or SQLDA2

You can develop most client applications using the standard SQLDA SQL descriptor area. However, you must use the extended SQLDA2 SQL descriptor area in the following situations:

- If your application needs to process data in columns that have a multibyte character data type.
- If your application needs the scale or precision of columns of type TIME, TIMESTAMP, or INTERVAL. This metadata information is not accessible if you use a standard SQLDA.

- If your application needs to access the full name of a column where the length of the column name is greater then 29 characters. If you use a standard SQLDA, Oracle SQL/Services truncates column numbers that are 30 or 31 characters long. The maximum length of a column name is 31 characters.

A

Deprecated and Obsolete Features

The following Oracle SQL/Services features have been deprecated or made obsolete. These features are no longer described in the main body of the *Guide to Using the Oracle SQL/Services Client API*, the *Oracle SQL/Services Installation Guide*, and the *Oracle SQL/Services Server Configuration Guide* and are described in detail only in this appendix. Oracle Corporation will not enhance features that are deprecated, and may announce in a future version of Oracle SQL/Services that these deprecated features are obsolete and can no longer be used. Therefore, applications that use these features should be modified accordingly when possible.

Deprecated Features

The following features have been deprecated.

- MS–DOS large memory model client API

  The MS–DOS large memory model client API is deprecated in Oracle SQL/Services V7.0, is frozen at the V6.1 level, and will not be enhanced in future releases. The MS–DOS large memory model client API was supplied with the V6.1 kit, but is no longer supplied with the V7.0 kit.

- MS–DOS medium memory model client API

  The MS–DOS medium memory model client API was deprecated in Oracle SQL/Services V5.1, is frozen at the V4.2 level, and will not be enhanced in future releases. The MS–DOS medium memory model client API was supplied with the V6.1 kit, but is no longer supplied with the V7.0 kit.

- ULTRIX for VAX client API

  The ULTRIX for VAX client API was deprecated in Oracle SQL/Services V5.1, is frozen at the V4.2 level, and will not be enhanced in future releases. The ULTRIX for VAX client API was supplied with the V6.1 kit, but is no longer supplied with the V7.0 kit.

- ULTRIX for RISC client API

  The ULTRIX for RISC client API is deprecated in Oracle SQL/Services V7.0, is frozen at the V6.1 level, and will not be enhanced in future releases. The ULTRIX for VAX client API was supplied with the V6.1 kit, but is no longer supplied with the V7.0 kit.

- SunOS client API

  The SunOS client API is deprecated in Oracle SQL/Services V7.0, is frozen at the V6.1 level, and will not be enhanced in future releases. The SunOS client API was supplied with the V6.1 kit, but is no longer supplied with the V7.0 kit.

- OS/2 client API

  The OS/2 client API was deprecated in Oracle SQL/Services V5.1, is frozen at the V4.2 level, and will not be enhanced in future releases. The V4.2 OS/2 client API was supplied with the V6.1 kit, but is no longer supplied with the V7.0 kit.

- VAX format of all API routines

  The VAX format for all API routines was deprecated in Oracle SQL/Services V5.0 and is documented only in this appendix. See Section A.1 for the VAX format syntax for these routines.

- Association routines

  - info_type parameter value SQLSRV_INFO_DB_CLASS of the sqlsrv_get_associate_info routine

    The info_type parameter value SQLSRV_INFO_DB_CLASS of the sqlsrv_get_associate_info routine is deprecated in V7.0. This parameter value will continue to work for Oracle SQL/Services V7.0. The SQLSRV_INFO_DB_CLASS parameter value describes a flag that returns true if the client is connected to a database service; the value is returned as a longword. This is useful to determine whether an attach is required after the association.

    Oracle Corporation recommends for V7.0 that you use the new info_type parameter value SQLSRV_INFO_SERVICE_ATTRS. See sqlsrv_get_associate_info for more information.

  - sqlsrv_set_server_class

    The sqlsrv_set_server_class routine was deprecated in Oracle SQL/Services V6.1. Oracle Corporation recommends that you use the class_name field in the association structure as the method of choosing a server class because this method works for multiassociation applications. See sqlsrv_set_server_class for a complete description of the sqlsrv_set_server_class routine.

- sqlsrv_set_transport_type

  The sqlsrv_set_transport_type routine was deprecated in Oracle SQL/Services V6.1. Oracle Corporation recommends that you use the xpttyp field in the association structure as the method of choosing a transport because this method works for multiassociation applications. See sqlsrv_set_transport_type for a complete description of the sqlsrv_set_transport_type routine.

- SQL statement routines
  - sqlsrv_execute–Execute prepared statement

    The routine sqlsrv_execute was deprecated in Oracle SQL/Services V7.0. Oracle Corporation recommends that you code your applications using the sqlsrv_execute_in_out routine. See sqlsrv_execute_in_out for a complete description of the sqlsrv_execute_in_out routine.

- Functional interface routines
  - sqlsrv_sqlda_map_data–Return column information

    The routine sqlsrv_sqlda_map_data was deprecated in Oracle SQL/Services V5.0. Oracle Corporation recommends that you code your applications using the sqlsrv_sqlda_ref_data routine. See sqlsrv_sqlda_ref_data or sqlsrv_sqlda2_ref_data for a complete description of the sqlsrv_sqlda_ref_data routine.

  - sqlsrv_sqlda_unmap_data–Free resources

    The routine sqlsrv_sqlda_unmap_data was deprecated in Oracle SQL/Services V5.0. Oracle Corporation recommends that you code your applications using the sqlsrv_sqlda_unref_data or sqlsrv_sqlda2_unref_data routine. See sqlsrv_sqlda_unref_data or sqlsrv_sqlda2_unref_data for a complete description of the sqlsrv_sqlda_unref_data routine.

Obsolete Features

An obsolete feature is a feature that is no longer supported that was described as a deprecated feature in a previous release. These features no longer work.

The following features have been made obsolete.

- Association routines
  - sqlsrv_get_environment—Return environment variable values

    The sqlsrv_get_environment routine was made obsolete in V6.1. This routine no longer exists.

  - sqlsrv_set_environment—Set environment variable values

    The sqlsrv_set_environment routine was made obsolete in V6.1. This routine no longer exists.

- sqlsrv_set_filter—Define filter for result table

  The sqlsrv_set_filter was made obsolete in Oracle SQL/Services V4.2. This routine no longer exists.

- Structures

  - SQLSRV_ENV_STR—Environment variable structure

    The SQLSRV_ENV_STR structure was made obsolete in V6.1. This structure no longer exists.

- Local Mode

  Local mode was made obsolete in Oracle SQL/Services V6.1. The local mode flag is ignored if it is set in the association structure.

A.1 VAX Format of Oracle SQL/Services API Routines

This section describes the deprecated VAX format syntax for all Oracle SQL/Services API routines.

A.1.1 VAX Format Section

Digital Equipment Corporation required that all callable products that run on the OpenVMS operating system have routine names in the format facility_name$routine_name. Thus, the VAX Format section of the template shows the routine name in the format SQLSRV$routine_name.

However, the dollar sign character ($) is not portable to all supported platforms. Some C compilers return a syntax error when they encounter a dollar sign character. Thus, Oracle SQL/Services automatically maps routine calls in the portable C format to the dollar sign format in a manner that is transparent to your application.

In the VAX Format section:

- The entry point name is shown in uppercase letters.
- The argument names are shown in lowercase letters.
- One or more spaces are used between the entry point name and the first argument, and between each argument.

- Brackets surround optional arguments. In Oracle SQL/Services, optional arguments cannot be omitted; a value of 0, passed by value, indicates that the API is to ignore the parameter.
- Commas precede arguments instead of following them.

VAX Format of Oracle SQL/Services API Routines

sqlsrv_abort—Disconnect Association

Format

```
SQLSRV$ABORT   associate_id
```

sqlsrv_allocate_sqlda_data or sqlsrv_allocate_sqlda2_data—Allocate Variables

Format

```
SQLSRV$ALLOCATE_SQLDA_DATA   associate_id ,sqlda_str
```

sqlsrv_associate—Create Client/Server Association

Format

```
SQLSRV$ASSOCIATE   node_name ,[user_name] ,[password] ,[read_buffer],[write_buffer],
                   [read_buffer_size] ,[write_buffer_size] ,[sqlca_str] ,associate_str ,associate_id
```

sqlsrv_close_cursor—Release Result Table

Format

```
SQLSRV$CLOSE_CURSOR   associate_id ,cursor_name
```

sqlsrv_declare_cursor—Declare a Cursor

Format

```
SQLSRV$DECLARE_CURSOR   associate_id ,cursor_name ,statement_id ,cursor_type ,cursor_mode
```

sqlsrv_execute—Execute Prepared Statement

Format

```
SQLSRV$EXECUTE   associate_id ,database_id ,statement_id ,execute_flag,
                 parameter_marker_sqlda,select_list_sqlda
```

sqlsrv_execute_in_out—Execute Prepared Statement

Format

```
SQLSRV$EXECUTE_IN_OUT   associate_id ,database_id ,sql_statement
```

sqlsrv_execute_immediate—Prepare and Execute Statement

Format

```
SQLSRV$EXECUTE_IMMEDIATE   associate_id ,database_id ,sql_statement
```

sqlsrv_fetch—Get Row from Result Table

Format

SQLSRV$FETCH associate_id ,cursor_name ,scroll_option ,position ,select_list_sqlda

sqlsrv_fetch_many—Get Multiple Rows from Result Table

Format

SQLSRV$FETCH_MANY associate_id ,cursor_name ,increment ,repeat_count

sqlsrv_free_sqlda_data or sqlsrv_free_sqlda2_data —Release Variables

Format

SQLSRV$FREE_SQLDA_DATA associate_id ,sqlda_str

sqlsrv_get_associate_info— Get Association Information

Format

SQLSRV$GET_ASSOCIATE_INFO associate_id ,info_type ,buf_len ,info_buf ,info_num

sqlsrv_open_cursor—Create Result Table

Format

SQLSRV$OPEN_CURSOR associate_id ,cursor_name ,statement_id ,parameter_marker_sqlda

sqlsrv_prepare—Compile Statement and Initialize SQLDA

Format

SQLSRV$PREPARE associate_id ,database_id ,sql_statement ,statement_id,
parameter_marker_sqlda ,select_list_sqlda

sqlsrv_release—Release Client/Server Association

Format

SQLSRV$RELEASE associate_id [,stats]

sqlsrv_release_statement—Release Statement Resources

Format

SQLSRV$RELEASE_STATEMENT associate_id ,statement_id_count ,statement_id_array

sqlsrv_set_server_class—Select Server Class

Format

SQLSRV$SET_SERVER_CLASS class_name

sqlsrv_set_transport_type—Select Network Transport Type

VAX Format of Oracle SQL/Services API Routines

Format

```
SQLSRV$SET_TRANSPORT_TYPE   transport_type
```

sqlsrv_sqlca_error—Return Error Codes

Format

```
SQLSRV$SQLCA_ERROR   associate_id ,major_error ,minor_error_first ,minor_error_second
```

sqlsrv_sqlca_error_text—Return Error Text

Format

```
SQLSRV$SQLCA_ERROR_TEXT   associate_id ,msglen ,text ,text_len
```

sqlsrv_sqlca_count—Return SQLCA.ERRD[2]

Format

```
SQLSRV$SQLCA_COUNT   associate_id
```

sqlsrv_sqlca_sqlerrd—Return Extended Information

Format

```
SQLSRV$SQLCA_SQLERRD   associate_id ,sqlerrd_array
```

sqlsrv_sqlda_sqld or sqlsrv_sqlda2_sqld—Return Number of Active Columns

Format

```
SQLSRV$SQLDA_SQLD   sqldaid
```

sqlsrv_sqlda_column_name or sqlsrv_sqlda2_column_name—Copy Column Name

Format

```
SQLSRV$SQLDA_COLUMN_NAME   sqldaid ,colnum ,colnam ,colnamlen
SQLSRV$SQLDA2_COLUMN_NAME   sqldaid ,colnum ,colnam ,colnamlen
```

sqlsrv_sqlda_column_type or sqlsrv_sqlda2_column_type—Return Column Type

Format

```
SQLSRV$SQLDA_COLUMN_TYPE   sqldaid ,colnum ,coltyp ,collen ,colscl ,rsv
SQLSRV$SQLDA2_COLUMN_TYPE   sqldaid ,colnum ,coltyp ,collen ,colscl ,coloctlen ,rsv
```

sqlsrv_sqlda_bind_data or sqlsrv_sqlda2_bind_data—Bind User Buffers to SQLDA and SQLDA2 Variables

Format

SQLSRV$SQLDA_BIND_DATA   sqldaid ,colnum ,coltyp ,collen ,colscl ,datptr ,nulptr ,rsv

SQLSRV$SQLDA2_BIND_DATA   sqldaid ,colnum ,coltyp ,collen ,colscl ,datptr ,nulptr ,octet_len,
chrono_scale ,chrono_precision ,rsv

sqlsrv_sqlda_unbind_sqlda or sqlsrv_sqlda2_unbind_sqlda2—Release SQLDA or SQLDA2 Variables

Format

SQLSRV$SQLDA_UNBIND_SQLDA   sqldaid

sqlsrv_sqlda_map_data—Return Column Information

Format

SQLSRV$SQLDA_MAP_DATA   sqldaid ,colnum ,coltyp ,collen ,colscl ,val ,nullp ,rsv

sqlsrv_sqlda_unmap_data—Free Resources

Format

SQLSRV$SQLDA_UNMAP_DATA   sqldaid ,colnum

sqlsrv_sqlda_ref_data or sqlsrv_sqlda2_ref_data—Return Column Information

Format

SQLSRV$SQLDA_REF_DATA   sqldaid ,colnum ,coltyp ,collen ,colscl ,val ,nullp ,rsv

SQLSRV$SQLDA2_REF_DATA   sqldaid ,colnum ,coltyp ,collen ,colscl ,val ,nullp ,octet_len,
chrono_scale, chrono_precision ,rsv

sqlsrv_sqlda_unref_data or sqlsrv_sqlda2_unref_data —Free Resources

Format

SQLSRV$SQLDA_UNREF_DATA   sqldaid ,colnum

sqlsrv_sqlda_get_data or sqlsrv_sqlda2_get_data—Copy Data from SQLDA and SQLDA2

Format

SQLSRV$SQLDA_GET_DATA   sqldaid ,colnum ,offset ,dst ,dstlen ,nullp ,bytcpy

SQLSRV$SQLDA2_GET_DATA   sqldaid ,colnum ,offset ,dst ,dstlen ,nullp ,bytcpy

sqlsrv_sqlda_set_data or sqlsrv_sqlda2_set_data—Copy Value of Column

Format

```
SQLSRV$SQLDA_SET_DATA   sqldaid ,colnum ,offset ,dst ,dstlen ,nullp ,bytcpy
```

```
SQLSRV$SQLDA2_SET_DATA   sqldaid ,colnum ,offset ,dst ,dstlen ,nullp ,bytcpy
```

sqlsrv_sqlda2_char_set_info—Return SQL Character Set Fields from SQLDA2

Format

```
SQLSRV$SQLDA2_CHAR_SET_INFO   sqldaid ,colnum ,name ,name_len ,schema ,schema_len,
                              catalog, catalog_len
```

sqlsrv_sqlda_set_sqllen or sqlsrv_sqlda2_set_sqllen—Set the SQLDA or SQLDA2 SQLLEN Field

Format

```
SQLSRV$SQLDA_SET_SQLLEN   sqldaid ,colnum ,len
```

```
SQLSRV$SQLDA2_SET_SQLLEN   sqldaid ,colnum ,len ,octet_len
```

sqlsrv_set_option—Sets the Option

Format

```
SQLSRV$SET_OPTION   association ,option ,value ,rsv
```

A.2 Deprecated Routines

The following routines are deprecated.

sqlsrv_execute

The sqlsrv_execute routine executes a prepared, executable SQL statement that does not return values in select list items.

C Format

```
extern int sqlsrv_execute(
                ASSOCIATE_ID associate_id,
                SQS_LONGWORD database_id,
                SQS_LONGWORD statement_id,
                short int execute_flag,
                SQLDA_ID parameter_marker_sqlda);
```

Parameters

associate_id

An identifier used to distinguish one active association from all others.

database_id

This parameter must be 0. Databases are referenced within the SQL statement syntax.

statement_id

The statement ID returned previously by sqlsrv_prepare when the statement was prepared. If you start batched execution for a particular statement ID using the SQLSRV_EXE_BATCH flag, you must end batched execution for that statement ID using one of the SQLSRV_EXE_W_DATA, SQLSRV_EXE_WO_DATA or SQLSRV_EXE_ABORT flags before you can execute any other prepared statement.

execute_flag

For a prepared statement that contains parameter markers, this parameter specifies whether the API sends single or multiple sets of parameter marker values to the server for processing (see Section 4.1 for more information on batched execution). For all other prepared SQL statements, this value must be 0 (SQLSRV_EXE_W_DATA). The values of the execute_flag parameter are shown in Table A–1.

Table A–1 Values of the execute_flag Parameter in sqlsrv_execute_in_out

Flag Name	Value	Description
SQLSRV_EXE_W_DATA	0	Builds an execute request message in the message buffer using the current values in the parameter marker SQLDA or SQLDA2, then sends the message to the server for execution. If batched execution is currently in effect for the statement, this parameter appends the new message to the previous messages in the message buffer, and sends all the messages to the server for execution along with any requests already queued at the server.
SQLSRV_EXE_BATCH	1	Starts or continues batched execution by building an execute request message in the message buffer using the current values in the parameter marker SQLDA or SQLDA2. If batched execution is already in effect for the statement, this parameter appends the new message to the previous messages in the message buffer. Using batched execution, no messages are sent to the server until the message buffer fills up, whereupon the messages in the message buffer are sent to the server to be queued up for subequent execution behind any previously queued requests.

(continued on next page)

Table A–1 (Cont.) Values of the execute_flag Parameter in sqlsrv_execute_in_out

Flag Name	Value	Description
SQLSRV_EXE_WO_DATA	2	Ends batched execution by sending the current contents of the message buffer to the server for execution along with any previously queued requests. Note that the current values in the parameter marker SQLDA or SQLDA2 are *not* sent to the server when batched execution is ended using the SQLSRV_EXE_WO_DATA flag.
SQLSRV_EXE_ABORT	3	Aborts batched execution by discarding the current contents of the message buffer and sending a message to the server directing it to discard any previously queued requests.

parameter_marker_sqlda

An SQLDA_ID that identifies the parameter marker SQLDA or SQLDA2 containing any parameter marker values or input and input/output argument values for the SQL statement to be executed.

Notes

- On successful completion of a call to sqlsrv_execute, Oracle SQL/Services stores the total number of database rows inserted, updated, or deleted in the SQLERRD[2] field of the SQLCA. Because multiple rows may be updated or deleted when you execute an UPDATE or DELETE statement, this value may be higher than the number of times that you called sqlsrv_execute_in_out for a particular batched execution. You can retrieve the row count from the SQLCA using the sqlsrv_sqlca_count routine. Note that Oracle Rdb does not return a row count value if you use the CALL statement to invoke a stored procedure, or if you execute a compound statement.

- If an error occurs executing a request queued for batched execution, then the server discards any remaining requests in the batch execution queue and returns the error to the client. Currently, there is no way to determine precisely which request caused the failure. Therefore, client applications will typically roll back the transaction in this situation.
- If you use batched execution to execute an SQL statement containing both parameter markers and select list items, such as UPDATE . . . RETURNING, then only the results from the execution of the last queued request are returned to the client. The results from the execution of all previously queued requests are lost.
- Once you start batched execution for a particular statement ID, you cannot call any API routines other than sqlsrv_execute_in_out, nor can you execute any other prepared statements until you end batched execution for the current statement ID using one of the SQLSRV_EXE_W_DATA, SQLSRV_EXE_WO_DATA, or SQLSRV_EXE_ABORT flags.

Errors

SQLSRV_CONNTIMEOUT	The connection to the server could not be completed within the specified time limit.
SQLSRV_EXEINTERR	The executor has encountered an internal or other error condition.
SQLSRV_DATA_TOO_LONG	The Oracle SQL/Services executor determined that the length of a data value in an SQLDA exceeded the maximum allowed for the value's data type.
SQLSRV_INTERR	Internal error.
SQLSRV_INVARG	Invalid routine parameter.
SQLSRV_INVASC	Invalid association identifier.
SQLSRV_INVEXEFLG	Invalid execute flag.
SQLSRV_INVSELLST	Invalid SQLDA or SQLDA2 select list.
SQLSRV_INVSQLDA	Invalid SQLDA, SQLDA2, or SQLDA_ID.
SQLSRV_INVSTMID	Invalid statement identifier.
SQLSRV_MULTI_ACT	A batched sqlsrv_execute_in_out or sqlsrv_fetch_many context is active.
SQLSRV_NETERR	Network transport returned an error.
SQLSRV_SVC_SHUTDOWN	The specified service has been shut down.

sqlsrv_set_server_class

The sqlsrv_set_server_class routine identifies the name of the service with which to associate. The sqlsrv_set_server_class routine must be called prior to calling the sqlsrv_associate routine for the service name to be passed to the server on the associate request. The association passes the most recently specified service name to the server.

Note

Oracle Corporation recommends that you use the class_name field in the association structure as the method of choosing a service name because this method works for multiassociation applications.

C Format

```
extern int sqlsrv_set_server_class(
               char *class_name );
```

Parameters

class_name

Address of a null-terminated string that identifies the name of the service with which to associate. A null class_name parameter clears the current service name. If you do not specify a service name, the API uses GENERIC.

Notes

None.

Errors

SQLSRV_INVARG	Invalid routine parameter.

sqlsrv_set_transport_type

The sqlsrv_set_transport_type routine allows you to set the type of network that you want an application to use when sending requests to the dispatcher. On most client systems, with Macintosh and Solaris being exceptions, if you do not call this routine, Oracle SQL/Services uses DECnet by default. For the Macintosh system, only if you do not use the sqlsrv_set_transport_type routine and do not set the transport with the Control Panel Device will Oracle SQL/Services use DECnet as the default. For the Solaris system, TCP/IP is always the default because that transport is the only one supported.

The association uses the most recently specified network type to connect to the dispatcher.

Note

Oracle Corporation recommends that you use the xpttyp field in the association structure as the method of choosing a transport because this method works for multiassociation applications.

C Format

```
extern int sqlsrv_set_transport_type(
                 int transport_type );
```

Parameters

transport_type

An integer value of SQLSRV_XPT_DECNET, SQLSRV_XPT_TCPIP, SQLSRV_XPT_ATK, or SQLSRV_XPT_SPXIPX that identifies the type of transport to use for communication between the client and server systems. Several rules apply to the specification of the transport type:

- SQLSRV_XPT_ATK is only allowed for Macintosh applications.
- SQLSRV_XPT_TCPIP is the only valid value for Solaris applications.
- SQLSRV_XPT_SPXIPX is only allowed for MS Windows V3.1 applications.

Notes

- The sqlsrv_set_transport_type routine overrides any transport setting made through the Macintosh Control Panel Device; however, if you want to use only the Control Panel Device for choosing a transport, do not call this routine in your application.
- If you do not use this call in your applications and you do not select a transport through the Macintosh Control Panel Device, Oracle SQL/Services uses DECnet by default for Macintosh applications.
- The value specified in the sqsapiw.ini or sqsapi32.ini file for transport type overrides the value specified for the transport_type parameter of the sqlsrv_set_transport_type routine.

Errors

SQLSRV_INVARG	Invalid routine parameter.

sqlsrv_sqlda_map_data

Note

The sqlsrv_sqlda_map_data routine was deprecated in Oracle SQL/Services Version 5.0. Oracle Corporation recommends that you not use the routine in the development of any Oracle SQL/Services applications. This routine may be removed in a future version.

The sqlsrv_sqlda_map_data routine returns the type, length, null value, and address of data for a column in the SQLDA.

C Format

```
extern int sqlsrv_sqlda_map_data(
                SQLDA_ID sqldaid,
                short int colnum,
                short int *coltyp,
                unsigned short int *collen,
                short int *colscl,
                PTRCHARPTR val,
                short int *nullp
                void *rsv);
```

Parameters

sqldaid

The identifier of a parameter marker or select list SQLDA or SQLDA2.

colnum

A column identified by its ordinal position in a parameter or select list.

coltyp

Address of a variable of type short into which the API writes the Oracle SQL/Services data type of the column.

collen

Address of a variable into which the API writes the length of the column. For an SQLDA, the column length is expressed as the number of 8-bit bytes and is stored as an unsigned word. For an SQLDA2, the column length is expressed as the number of characters, where a single character might occupy more than one byte in a multibyte character set, and is stored as an unsigned longword.

colscl

Address of a variable of type short into which the API writes the scale factor for columns of type SQLSRV_GENERALIZED_NUMBER or the type of date or interval for columns of type SQLSRV_GENERALIZED_DATE or SQLSRV_INTERVAL, respectively. Undefined for columns of all other data types.

val

The address of a variable of type CHARPTR into which the API writes the address of the column's data variable.

nullp

Address of a variable of type short into which the API writes the value of the column's indicator variable.

rsv

Argument reserved for future use. The value of this argument must be NULL.

Notes

- Oracle SQL/Services returns an error if the SQLDA or SQLDA2 is invalid or if the column number is greater than the number of parameter markers or select list items (colnum >= sqlda.SQLD).
- Use the sqlsrv_sqlda_map_data routine with the sqlsrv_allocate_sqlda_data or sqlsrv_allocate_sqlda2_data routine. It is equivalent to reading the SQLLEN, SQLTYPE, SQLDATA, and SQLINDA fields of the SQLVARARY array for the column.
- Oracle SQL/Services transparently allocates memory for data and indicator variables using handles on the Macintosh platform. To provide a consistent, portable interface on the Macintosh platform with other client platforms, the sqlsrv_sqlda_ref_data and sqlsrv_sqlda2_ref_data routines lock the memory handles, then return the pointers, rather than the handles, to the data and indicator variables. Therefore, when your application has finished accessing data and indicator variables, you must call the sqlsrv_sqlda_

unref_data and sqlsrv_sqlda2_unref_data routines to unlock the memory handles.

Errors

SQLSRV_INVCOLNUM	Column number not within range.
SQLSRV_INVSQLDA	Invalid SQLDA, SQLDA2, or SQLDA_ID.

sqlsrv_sqlda_unmap_data

Note

The sqlsrv_sqlda_unmap_data routine was deprecated in Oracle SQL/Services Version 5.0. Oracle Corporation recommends that you not use the routine in the development of any Oracle SQL/Services applications. This routine may be removed in a future version.

The sqlsrv_sqlda_unmap_data routine frees any unwanted resources tied up by the sqlsrv_sqlda_map_data routine.

Note

The sqlsrv_sqlda_unmap_data routine can be used with either the SQLDA or SQLDA2.

C Format

```
extern int sqlsrv_sqlda_unmap_data(
                SQLDA_ID sqldaid,
                short int colnum);
```

Parameters

sqldaid

The identifier of a parameter marker or select list SQLDA or SQLDA2.

colnum

A column identified by its ordinal position in a parameter or select list.

sqlsrv_sqlda_unmap_data

Notes

- Oracle SQL/Services returns an error if the SQLDA or SQLDA2 is invalid or if the column number is greater than the number of parameter markers or select list items (colnum >= sqlda.SQLD).
- Oracle SQL/Services transparently allocates memory for data and indicator variables using handles on the Macintosh platform. To provide a consistent, portable interface on the Macintosh platform with other client platforms, the sqlsrv_sqlda_ref_data and sqlsrv_sqlda2_ref_data routines lock the memory handles, then return the pointers, rather than the handles, to the data and indicator variables. Therefore, when your application has finished accessing data and indicator variables, you must call the sqlsrv_sqlda_unref_data and sqlsrv_sqlda2_unref_data routines to unlock the memory handles.

Errors

SQLSRV_INVCOLNUM	Column number not within range.
SQLSRV_INVSQLDA	Invalid SQLDA, SQLDA2, or SQLDA_ID.

Index