



IDL DataMiner Guide



IDL Version 5.3
Sept., 1999 Edition
Copyright © Research Systems, Inc.
All Rights Reserved

Restricted Rights Notice

The IDL[®] software program and the accompanying procedures, functions, and documentation described herein are sold under license agreement. Their use, duplication, and disclosure are subject to the restrictions stated in the license agreement. Research Systems, Inc., reserves the right to make changes to this document at any time and without notice.

Limitation of Warranty

Research Systems, Inc. makes no warranties, either express or implied, as to any matter not expressly set forth in the license agreement, including without limitation the condition of the software, merchantability, or fitness for any particular purpose.

Research Systems, Inc. shall not be liable for any direct, consequential, or other damages suffered by the Licensee or any others resulting from use of the IDL software package or its documentation.

Permission to Reproduce this Manual

If you are a licensed user of this product, Research Systems, Inc. grants you a limited, nontransferable license to reproduce this particular document provided such copies are for your use only and are not sold or distributed to third parties. All such copies must contain the title page and this notice page in their entirety.

Acknowledgments

IDL[®] is a trademark of Research Systems Inc., registered in the United States Patent and Trademark Office, for the computer program described herein.

Numerical Recipes[™] is a trademark of Numerical Recipes Software. Numerical Recipes routines are used by permission.

GRG2[™] is a trademark of Windward Technologies, Inc. The GRG2 software for nonlinear optimization is used by permission.

NCSA Hierarchical Data Format (HDF) Software Library and Utilities
Copyright 1988-1998 The Board of Trustees of the University of Illinois
All rights reserved.

Portions of this software are copyrighted by INTERSOLV, Inc., 1991-1998.

Other trademarks and registered trademarks are the property of the respective trademark holders.



Research Systems, Inc. documentation is printed on recycled paper. Our paper has a minimum 20% post-consumer waste content and meets all EPA guidelines.



Contents

Chapter 1:	
Overview	9
Introduction to IDL DataMiner and ODBC	10
What is IDL DataMiner?	10
What is the ODBC?	10
ODBC Conformance Levels	12
API Conformance Levels	12
SQL Conformance Levels	12
Where to Find Additional Information	14
About this Volume	15
Audience	15
Organization	15

Conventions	16
Terminology	16
Network Access Requirements	17
Installation on UNIX Systems	18
Initialization	19
Mappings	20
Error Messages	21
Chapter 2:	
Using the IDL DataMiner	23
Components	24
Using the DB_EXISTS Function	25
Creating a Database Object	26
Finding Available Databases	26
Finding a Specific Database	26
Connecting to a Database	27
Finding Tables	31
Finding Available Tables	31
Finding Specific Tables	31
Connecting to a Table	32
Working with Table Data	33
Moving through a Recordset	33
Example	35
ODBC SQL Syntax Notes	37
Date, Time, and Timestamp Data	37
Scalar Functions	37
LIKE Predicate Escape Characters	38
Outer Joins	38
Procedure Calls	38

Chapter 3:	
IDL DataMiner API	41
How to Use this Chapter	42
Creating Database Objects	44
Destroying Database Objects	44
DIALOG_DBCONNECT()	45
DB_EXISTS()	46
IDLdbDatabase	47
IDLdbDatabase::Connect	48
IDLdbDatabase::ExecuteSQL	49
IDLdbDatabase::GetDatasources	50
IDLdbDatabase::GetProperty	51
IDLdbDatabase::GetTables	53
IDLdbDatabase::SetProperty	54
IDLdbRecordset	56
IDLdbRecordset::AddRecord	58
IDLdbRecordset::CurrentRecord	59
IDLdbRecordset::DeleteRecord	60
IDLdbRecordset::GetField	61
IDLdbRecordset::GetProperty	62
IDLdbRecordset::GetRecord	65
IDLdbRecordset::MoveCursor	66
IDLdbRecordset::NFields	67
IDLdbRecordset::SetField	68
Chapter 4:	
Understanding the ODBC.INI File	69
Overview	70
ODBC.INI File Format	71
ODBC Data Sources	71
Data Source Specification	71
Default Data Source Specification	72

ODBC Options	72
ODBC.INI File Example	74
Chapter 5:	
Using Intersolv ODBC Drivers	75
Supported Drivers	76
Connect ODBC for INFORMIX	79
System Requirements	79
Configuring Data Sources	81
Connecting to a Data Source Using a Logon Dialog Box	85
Connecting to a Data Source Using a Connection String	86
Data Types	89
Isolation and Lock Levels Supported	92
ODBC Conformance Level	92
Number of Connections and Statements Supported	92
Connect ODBC for Oracle	93
System Requirements	93
Configuring Data Source	95
Connecting to a Data Source Using a Logon Dialog Box	100
Connecting to a Data Source Using a Connection String	100
Stored Procedure Results	107
Isolation and Lock Levels Supported	107
ODBC Conformance Level	107
Number of Connections and Statements Supported	108
Connect ODBC for Sybase	109
System Requirements	109
Configuring Data Sources	110
Connecting to a Data Source Using a Logon Dialog Box	118
Connecting to a Data Source Using a Connection String	119
Data Types	124
Isolation and Lock Levels Supported	125
ODBC Conformance Level	125

Number of Connections and Statements Supported	126
Connect ODBC for Text	127
System Requirements	127
Formats for Text Files	127
Configuring Data Sources	128
Defining Table Structure	135
Defining Table Structure on UNIX Platforms	138
Date Masks	140
Connecting to a Data Source Using a Connection String	142
Data Types	147
Select Statement	147
Alter Table Statement	148
ODBC Conformance Level	148
Number of Connections and Statements Supported	148
The UNIX Environment	149
The System Information File (.odbc.ini)	149
Optional Environment Variables	151
Using Double-Byte Character Sets	151
Translators	151
Locking and Isolation Levels	152
Locking	152
Isolation Levels	152
Locking Modes and Levels	155
Chapter 6:	
ODBC API and Scalar Functions	157
API Functions	158
Scalar Functions	161
Index	171



Chapter 1: Overview

The following topics are discussed in this chapter:

Introduction to IDL DataMiner and ODBC	10	Network Access Requirements	17
ODBC Conformance Levels	12	Installation on UNIX Systems	18
Where to Find Additional Information	14	Initialization	19
About this Volume	15	Mappings	20
Conventions	16	Error Messages	21

Introduction to IDL DataMiner and ODBC

The IDL DataMiner is an Open Database Connectivity (ODBC) interface that allows IDL users to access and manipulate information from a variety of database management systems. Research Systems, Inc., developed IDL DataMiner so that IDL users can have all the connectivity advantages of ODBC without having to understand the intricacies of ODBC or SQL (Structured Query Language).

What is IDL DataMiner?

IDL DataMiner is a database-independent API for accessing and manipulating data in IDL. The IDL DataMiner allows you to perform actions including the following:

- Connect to a database management system (DBMS)
- Query data from a DBMS
- Get information about the available database tables in a DBMS
- Access a table in a DBMS
- Create a table in a DBMS
- Delete a table in a DBMS
- Perform standard SQL operations in the DBMS
- Get information about the columns in a selected table
- Add/Change/Delete records in a table

What is the ODBC?

ODBC stands for Open Database Connectivity, an interface that allows applications to access data in database management systems (DBMSs) using Structured Query Language (SQL) as a standard for accessing data.

SQL is ODBC's standard for accessing data and is a widely accepted industry standard for data definition, data manipulation, data management, access protection, and transaction control. The IDL DataMiner was designed so that users would not be required to have a knowledge of SQL to access data sources. However, DataMiner *does* provide an execution routine which allows users to perform any valid SQL statement (including creating, retrieving, and deleting tables in a database).

The ODBC specification defines a vendor-independent API for accessing data stored in relational and non-relational databases. The Core functions and SQL grammar are

based on work done by the X/Open SQL Access Group. The ODBC architecture is made up of four components:

- **Database Application.** The database application calls functions defined in the ODBC API to access a data source.
- **Driver Manager.** The Driver Manager implements the ODBC API and provides information to an application—such as a list of available data sources and drivers—loads drivers dynamically as they are needed, and provides argument and state transition checking.
- **Drivers.** Each driver processes ODBC function calls and manages exchanges between an application and a data source.
- **Data Source.** A data source contains the data that an application needs to access. The data source includes the data, the database management system (DBMS) in which the data is stored, the platform on which the DBMS resides, and the network (if any) used to access the DBMS.

An *ODBC-compliant driver* allows you to communicate between an ODBC-compliant application and a DBMS. For example, the SYBASE SQL Server 10 driver allows you to connect your ODBC-compliant application to a Sybase SQL Server 10 database.

An ODBC driver is available on most major platforms. The information in the initialization file that the drivers use, the functions and SQL grammar that the drivers support, and the error message formats are the same across all platforms.

The ODBC DriverSet is made up of two ODBC components—the Driver Manager and a set of database drivers. With the ODBC DriverSet, you can access, query, and update data in a number of different databases.

ODBC Conformance Levels

ODBC defines two different conformance standards for drivers—the API conformance standard and the SQL conformance standard. Each conformance standard is made up of three levels. These levels help application and driver developers establish standard sets of functionality. See [Chapter 6, “ODBC API and Scalar Functions”](#) for more information on ODBC conformance levels.

API Conformance Levels

The API conformance standard is made up of three levels:

- **Core API.** A set of core functions that correspond to the functions in the X/Open SQL Access Group Call Level Interface specification.
- **Level 1 API.** Core API functionality plus all Level 1 functionality.
- **Level 2 API.** Core and Level 1 API functionality plus all Level 2 functionality.

ODBC API Functions

The Intersolv drivers support all Core and Level 1 functions. In addition, each driver supports a key set of the Level 2 functions. For a list of supported Level 2 functions by driver, refer to the “ODBC Conformance Levels” section in the chapter of the *Intersolv ODBC DriverSet Reference Chapter*.

SQL Conformance Levels

SQL conformance is made up of three levels—Minimum, Core, and Extended. The Minimum level is designed to meet the basic level of ODBC conformance. The Core level roughly corresponds to the X/Open SQL Access Group SQL CAE specification (1995) and the Extended level provides common DBMS extensions to SQL. Most of the Intersolv drivers support all Minimum and Core SQL grammar. In addition, each driver supports a number of extended SQL statements, expressions, and data types. For a list of supported Extended SQL grammar by driver, refer to the appropriate “ODBC Conformance Levels” section in each chapter.

Minimum SQL Grammar

The Minimum level of SQL grammar consists of the following statements, expressions, and data types:

- Data Definition Language (DDL): CREATE TABLE and DROP TABLE

- Data Manipulation Language (DML): simple SELECT, INSERT, UPDATE, SEARCHED, and DELETE SEARCHED
- Expressions: simple (such as $A > B + C$)
- Data types: CHAR, VARCHAR, or LONG VARCHAR

Core SQL Grammar

The Core level of SQL grammar consists of the following statements, expressions, and data types:

- Minimum SQL grammar and data types
- Data Definition Language (DDL): ALTER TABLE, CREATE INDEX, DROP INDEX, CREATE VIEW, DROP VIEW, GRANT, and REVOKE
- Data Manipulation Language (DML): full SELECT
- Expressions: subquery, set functions such as SUM and MIN
- Data Types: DECIMAL, NUMERIC, SMALLINT, INTEGER, REAL, FLOAT, DOUBLE PRECISION

Extended SQL Grammar

The Extended level of SQL grammar consists of the following statements, expressions, and data types:

- Minimum and Core SQL grammar and data types
- Data Manipulation Language (DML): outer joins, positioned UPDATE, positioned DELETE, SELECT FOR UPDATE, and unions
- Expressions: scalar functions such as SUBSTRING, ABS, date, time, and timestamp literals
- Data types: BIT, TINYINT, BIGINT, BINARY, VARBINARY, LONG VARBINARY, DATE, TIME, TIMESTAMP
- Batch SQL statements
- Procedure calls

Where to Find Additional Information

For more information about ODBC, refer to the following:

- **Microsoft ODBC Programmer's Reference and SDK Guide (Version 3.0).**
This programmer's reference introduces the ODBC architecture and explains how to write ODBC drivers and applications that use ODBC for Windows. It also contains the ODBC API Reference, in which each of the functions in the ODBC API is listed in alphabetic order and described in detail. The SDK guide explains how to install and use the SDK software.

About this Volume

The *IDL DataMiner Guide* describes IDL's ODBC interface as well as information about the specific ODBC drivers that are provided with the DataMiner system.

Audience

This manual assumes you have:

- a working knowledge of IDL,
- knowledge of your own DBMS.

Familiarity with SQL is helpful, but not required.

Organization

The *IDL DataMiner Guide* is divided into the following chapters:

- Chapter 1, (this chapter) discusses the manual's intended audience, organization of the manual, conventions, and lists other sources of information about ODBC.
- [Chapter 2, "Using the IDL DataMiner"](#), discusses IDL DataMiner functionality.
- [Chapter 3, "IDL DataMiner API"](#), is a reference explaining the IDL DataMiner object classes and their use.
- [Chapter 4, "Understanding the ODBC.INI File"](#), explains the ODBC initialization file and its contents.
- [Chapter 5, "Using Intersolv ODBC Drivers"](#), explains how to install and set up Intersolv Drivers.
- [Chapter 6, "ODBC API and Scalar Functions"](#), introduces the ODBC DriverSet, explains the API and SQL grammar conformance levels, and provides information about how to access translation libraries with the drivers.

Conventions

This section describes the conventions used in this manual to identify technical terms and computer language constructs.

Terminology

The following are terms that are used throughout this manual:

DBMS: Database Management System

data source: A specific instance of a combination of a DBMS product, any remote operating system, and network necessary to access the DBMS.

recordset: A subset of the records in the current database. Recordsets are created either by formulating an SQL query to select records or by selecting an existing named table in the database.

cursor: The current location or current record in a recordset.

Network Access Requirements

To access an external database, you must be able to connect to the network, have access to the external database, and have access to the server on which the external database is located. Database permissions are established using the security features of the external database. If you do not have the proper access permissions, consult your local database administrator.

Note

Some database systems require that a database-specific network package be installed. Consult your database and database driver documentation for details.

Installation on UNIX Systems

The DataMiner system components are installed by the IDL installation program (assuming you elected to install the DataMiner components when installing IDL). On some UNIX systems, the Oracle ODBC drivers must be linked against portions of the Oracle installation. For more information on how this is performed consult the files located in the Dataminer directory:

```
$IDL_DIR/bin/bin.<OS Name>/dm/src/oracle
```

where *<OS Name>* is the name of your operating system.

Note

If this directory does not exist, this operation is not required.

During the IDL installation process, an ODBC initialization file is created. This file describes which ODBC drivers are installed and allows for the setting of attributes of a driver. This generated file should be copied to the users home directory to allow the proper operation of the Dataminer system.

```
% cp $IDL_DIR/resource/dm/<OS Name>/odbc.ini ~/.odbc.ini
```

The `odbc.ini` file is used by the `odbc` system to determine what ODBC drivers are installed. You might have to edit this file to take into account any driver-specific information fields. See [Chapter 4, “Understanding the ODBC.INI File”](#) for details.

Initialization

When you install IDL and the ODBC database drivers, default values are written into the ODBC initialization file. Your particular environment might require you to change entries or you might wish to change entries so that you only have to enter login and password information once. The IDL DataMiner allows you to enter common database sources and parameters for those databases in the initialization file. When you change your initialization settings, you must restart IDL DataMiner for the changes to take effect. See [Chapter 4, “Understanding the ODBC.INI File”](#), for details.

Mappings

SQL data types have been mapped to IDL DataMiner data types so that you can access and manipulate the data without having to fully understand SQL. [Table 1-1](#) details these mappings.

IDL Type	SQL Type
STRING	DECIMAL
	NUMERIC
	CHAR
	LONG VARCHAR
BYTE	BIT
	TINYINT
	BIGINT
INT	SMALLINT
LONG	INTEGER
LONG64	BIGINT
FLOAT	REAL
DOUBLE	FLOAT
	DOUBLE PRECISION
BYTE ARRAY	BINARY
	VARBINARY
	VARCHAR
	LONG VARBINARY
ODBC_SQL_DATE Struct	DATE
ODBC_SQL_TIME Struct	TIME
ODBC_SQL_TIMESTAMP Struct	TIMESTAMP

Table 1-1: IDL - SQL Type Code Mapping

Error Messages

The error messages returned follow the ODBC standard error message format as outlined in the ODBC Software Development Kit. ODBC error messages use one of the following formats, depending on whether the VERBOSE property is set on the database. (See “[IDLdbDatabase::SetProperty](#)” on page 54 for a description of the VERBOSE property.)

Standard Messages

If the VERBOSE property is set equal to zero (the default), the error message has the form:

```
[vendor-identifier] [ODBC-component-identifier]
[data-source-identifier] data-source-text, component-text
```

where

[*vendor-identifier*] shows the vendor of the component in which the error occurred, or that received the error directly from the data source.

[*ODBC-component-identifier*] shows the component in which the error occurred, or that received the error directly from the data source.

[*data-source-identifier*] shows the data source in which the error occurred.

data-source-text is the text generated by the data source, if the error occurred in a data source.

component-text is the text generated by the ODBC component, if the error occurred in an ODBC component.

Verbose Messages

If the VERBOSE property is set equal to one, the following fields precede the standard error message:

```
SQL Function=<function name>, STATE=<state number>,
CODE=<error code>
```

where

<*function name*> is the actual ODBC C function that triggered the error. This information is needed for the interpretation of the STATE error code.

<*state number*> is a 5 character string that defines an error code returned from the odbc system. This code along with the SQL Function name can be used to determine the actual cause of the ODBC error.

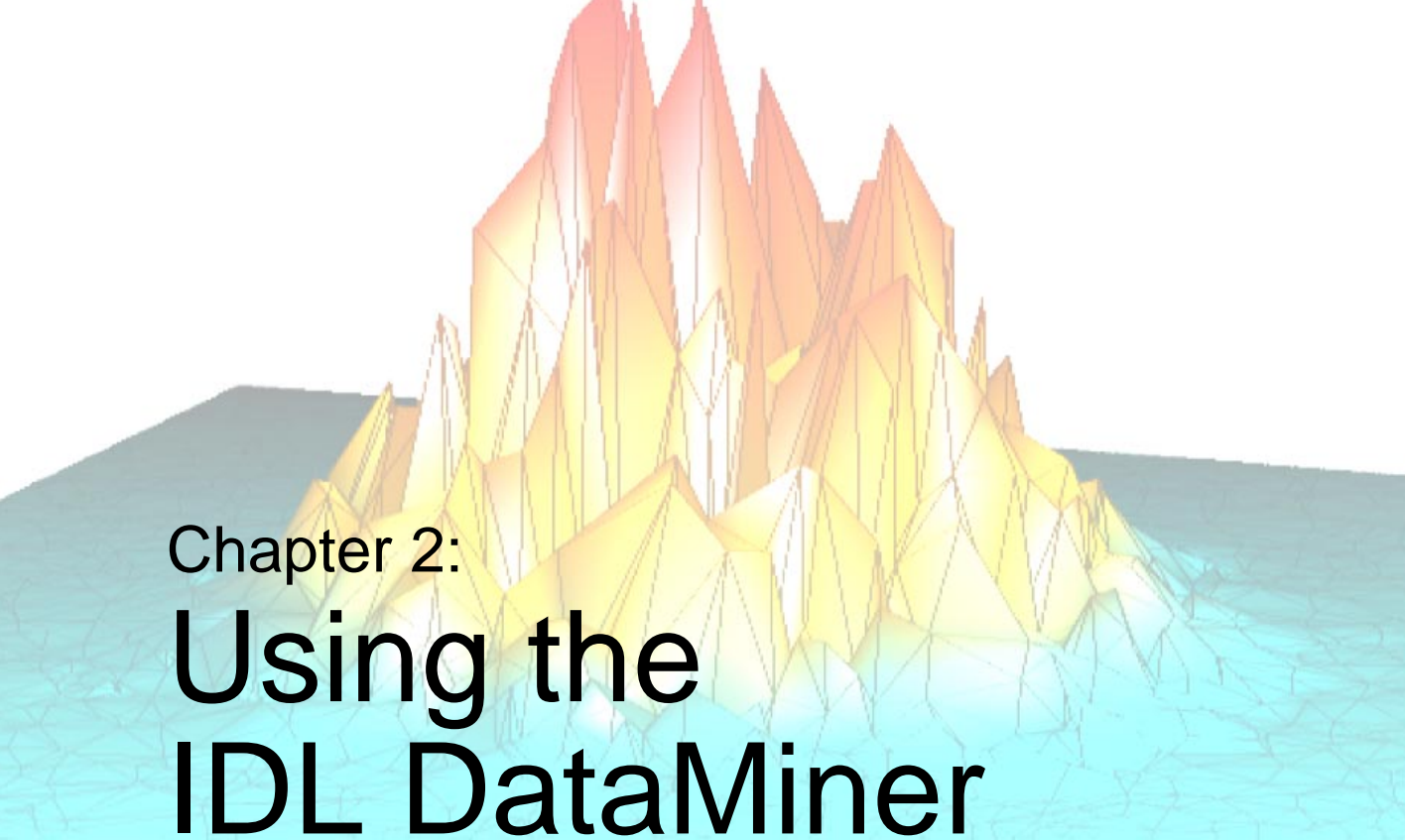
<error code> is the error code returned from the data source. This code is a native error of the datasource and describes the error condition that the datasource detected.

For example, a standard error message from a data source might look like this:

```
% IDLDBDATABASE::CONNECT: ODBC [Microsoft]
    [ODBC SQL Server Driver][netlibtcp]
    ConnectionOpen (connect()).
```

The verbose error message for the same error:

```
% IDLDBDATABASE::CONNECT: ODBC
    SQL Function=SQLDriverConnect, STATE=01000,
    CODE=146,[Microsoft][ODBC SQL Server Driver]
    [netlibtcp] ConnectionOpen (connect()).
```



Chapter 2:

Using the IDL DataMiner

This chapter describes the functionality and syntax of the IDL DataMiner. For more detail on how to use IDL DataMiner classes to perform actions on a DBMS, see [Chapter 3, “IDL DataMiner API”](#). For information on IDL commands and syntax, see the *IDL Reference Guide*.

Components	24	Connecting to a Table	32
Using the DB_EXISTS Function	25	Working with Table Data	33
Creating a Database Object	26	Example	35
Connecting to a Database	27	ODBC SQL Syntax Notes	37
Finding Tables	31		

Components

The IDL DataMiner provides two IDL objects for accessing databases:

- Database object (IDLdbDatabase)
- Recordset object (IDLdbRecordset)

A full discussion of IDL objects is beyond the scope of this manual. Consult the *Building IDL Applications* manual for details about IDL's object-oriented programming features.

The Database object contains instance data and methods that you can use to connect to, disconnect from, and perform operations on a DBMS. The Recordset object contains a database table or the results from a SQL query. You can use Recordset methods to manipulate table data.

Note

Some of the methods associated with IDL database and recordset objects are driver-dependent. This means that some functions are not available when you are using database drivers that do not support those functions.

The IDL DataMiner also provides an IDL function, `DIALOG_DBCONNECT`, that you can use to connect to the DBMS via the standard ODBC dialog boxes. Using this method, you are prompted for any information that is required to connect to the desired database.

Finally, the IDL function `DB_EXISTS` allows you to determine if database functionality is available and licensed on a specific platform.

Using the DB_EXISTS Function

The ODBC system is not available on all systems. Use the `DB_EXISTS()` function to determine if a database is available and licensed on your system. To check whether ODBC is available on your system, enter the following at the IDL prompt:

```
status = DB_EXISTS()
```

If IDL DataMiner and ODBC drivers are installed on your system and the DataMiner option is properly licensed, the `DB_EXISTS` function returns 1; otherwise the function returns 0.

Creating a Database Object

To connect to a database, you must first create an IDL Database Object using the following statement:

```
objDB = OBJ_NEW( 'IDLdbDatabase' )
```

The newly-created database object represents a connection to a datasource. The object is not considered valid until a connection to the datasource is made, either via the `Connect` method of the IDL Database Object or the `DIALOG_DBCONNECT` function.

Once the Database Object has been created, you can perform operations including:

- connecting to a database,
- finding out which databases are available,
- finding out if a specific database is available,
- get properties of the database object.

Finding Available Databases

To find out which databases are available, use the database object's `GetDatasources` method as follows.

```
sources = objDB->GetDatasources()
```

The result is an array of IDL structures containing the datasource names and descriptions of all available data sources. See [“IDLdbDatabase::GetDatasources”](#) on page 50 for further information on this structure.

Finding a Specific Database

To find out if a specific database is available, inspect the list of datasources returned by the `GetDatasources` method. The following IDL commands check to see if “Informix” is listed in the array of data sources, and if so, print the word “Yes” to the IDL command log:

```
index = WHERE(sources.datasource EQ 'Informix', nmatch)  
IF(nmatch ge 1) THEN PRINT, 'Yes'
```

If the desired database is reported as available, the database driver is installed on your system. You will still need to make sure that the driver is configured correctly before you are able to connect to a database.

Connecting to a Database

Once you have created a Database object, you can connect to a database. IDL DataMiner offers two options for accessing databases:

1. The `DIALOG_DBCONNECT` function and the ODBC dialog boxes.
2. The `connect` method of the IDL Database Object.

Connecting with the `DIALOG_DBCONNECT` Function

`DIALOG_DBCONNECT` is a function used to connect to a database using ODBC dialog boxes. These dialogs prompt you for information required to connect to the desired database.

To connect to a database using the `DIALOG_DBCONNECT` function, enter the following at the IDL prompt:

```
status = DIALOG_DBCONNECT(objDB)
```

The SQL Data Sources dialog box appears. This dialog box lists the currently defined Data Sources; it looks something like the following figure:

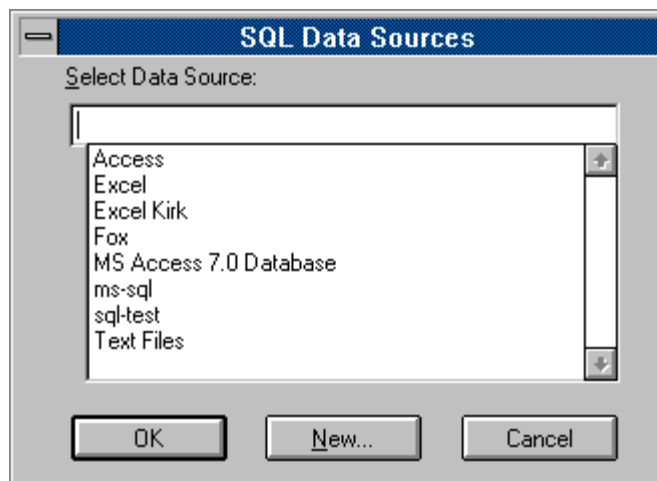


Figure 2-1: SQL Data Sources (Windows dialog)

You can take one of three actions:

Note

Due to Motif library inconsistencies, this dialog may fail on some UNIX systems.

- Select the desired data source and click “OK”. After selecting this button, a “true” value is returned if the database object connects to the data source.
- Cancel the operation by clicking “Cancel”. After selecting this button, a “false” value is returned, and the database object does not connect to the data source.
- On Windows systems, click “New” to define a new data source for the system. After selecting this button, the Add Data Source dialog box appears. This button is not available on Motif systems.

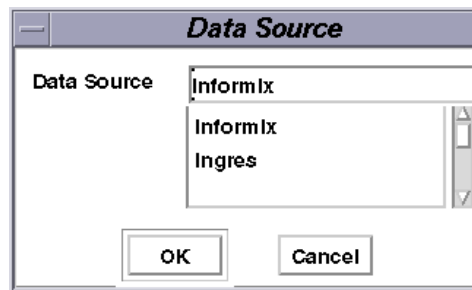


Figure 2-2: SQL Data Sources (Motif dialog)

Define a new data source for the system by selecting the desired ODBC driver from the list and clicking the OK button. The dialog box will close and you will be connected to a database. In some cases, you will see an additional configuration dialog after the Add Data Source dialog closes.

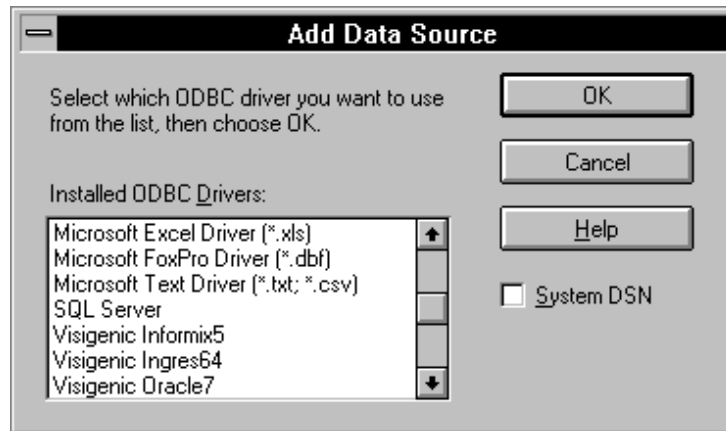


Figure 2-3: Add Data Source

Connecting with the IDL Database Object's Connect Method

To connect to a database using the database object's `Connect` method, enter the following at the IDL prompt:

```
objDB->Connect, datasource = source_name
```

where `source_name` is the name of the data source. One way to specify the `datasource` name is to provide an index into the array of `datasource` names created with the IDL commands shown in “[Finding Available Databases](#)” on page 26, above. For example, if you wanted to connect to the first available `datasource` in the list of available sources, you might use the following IDL commands:

```
sources = objDB->GetDatasources()
mysource = sources[0].datasource
objDB->Connect, datasource = mysource
```

Once you have connected to a database, you can perform several operations using IDL DataMiner methods. These operations include:

- finding out which tables are available in the `datasource`;
- finding specific tables in the `datasource`;
- executing SQL statements to perform actions such as creating a table or deleting a table;

- getting database properties;
- creating tables;
- creating a recordset and connecting to tables; and
- retrieving and manipulating table data.

Finding Tables

Once you have connected to the database, you can get a list of available tables or find a specific table.

Finding Available Tables

To find out which tables are available, use the `GetTables` method of the database object:

```
tables = objDB->GetTables()
```

Note

The `GetTables` method is not available with all drivers.

The result is an array of IDL structures containing information about the available tables. See “[IDLdbDatabase::GetTables](#)” on page 53 for further information on this structure.

Finding Specific Tables

To find out if a specific table is available, inspect the list of tables returned by the `GetTables` method. The following IDL commands check to see if “mytable” is listed in the array of tables, and if so, print the word “Yes” to the IDL command log:

```
index = WHERE(tables.name EQ 'mytable', nmatch)
IF(nmatch ge 1) THEN PRINT, 'Yes'
```

You are now ready to connect to the table and retrieve data.

Connecting to a Table

Connecting to a table and retrieving its data involves:

- creating a Recordset object,
- specifying the table from which the information is being retrieved.

The recordset object contains a database table or a selection based on criteria you specify in an SQL query. This object allows you to programmatically manipulate the data in the database. To create this object, a valid database object is required.

In the following example, a new Recordset object is being created for a table called “mydata.”

```
objRS = OBJ_NEW('IDLDBRecordset', objDB, table='mydata')
```

Once you have connected to a table, you can use IDL DataMiner methods to manipulate the data in several ways as depicted in the examples provided in the next section.

Note

When a table is selected, the entire data contained in the table is not automatically imported into IDL. This preserves memory. You can retrieve the desired data in a recordset by moving the cursor to the desired record via the `MoveCursor` method and then importing that data into IDL using the `GetField` or `GetRecord` method.

Working with Table Data

Once you have created the Recordset object and connected to a table, you can use IDL DataMiner methods to retrieve and manipulate the data in several ways. For example, you can:

- find out if a table is “ReadOnly”,
- get properties of the recordset,
- move the cursor,
- add records,
- delete records,
- retrieve fields,
- set fields,
- find the current row number in a recordset,
- find the number of fields in a recordset,
- get an array of field information structures, one for each field in the recordset.

You can also obtain information about a database or recordset concerning the following:

- the number of table fields,
- the name of DBMS associated with a database object,
- the DBMS version,
- a list of available drivers,
- the ODBC driver level,
- the ODBC driver version,
- the maximum number of connections.

Moving through a Recordset

Moving through recordsets is based on the concept of the *cursor*. The cursor is the current row, or record, in the recordset. When you refer to fields in a Recordset, you obtain values from the current record, and only the current record can be modified.

You can use the Recordset object's `MoveCursor` method to navigate through the records in a recordset. Keywords to the `MoveCursor` method allow you to specify new cursor locations.

In the following example, the `MoveCursor` method and `FIRST` keyword move to the first record.

```
status = objRS->MoveCursor(/FIRST)
```

In the following example, the `MoveCursor` method and `NEXT` keyword move to the next record.

```
status = objRS->MoveCursor(/NEXT)
```

Example

The following example steps you through the process of creating a database object, connecting to a datasource, creating a table, and moving data between the database and IDL. The example uses the SQLAnywhere server; you will need to replace references to the SQLAnywhere server with references to your own specific database server. In order to work through this example, you will need to be able to connect to and log on to your database server.

First, create a database object. Enter the following at the IDL command prompt:

```
oDB = obj_new('IDLDBDatabase')
```

Use the GetDatasources method to discover the names of the datasources available on your system, then print the list to your command log window:

```
sources = oDB->GetDatasources()  
PRINT, sources.datasource, FORMAT='(a)'
```

IDL will print something like the following:

```
SybaseDBLib  
Sybase  
SQLAnywhere  
Oracle  
Ingres  
Informix  
MSSQLServer
```

Connect to the SQLAnywhere server. (Substitute your own datasource, username, and password.)

```
oDB->Connect, DataSource = 'SQLAnywhere', $  
user=username, password=passwd
```

Get a list of the available tables:

```
tables = oDB->GetTables()  
PRINT, atables.name, FORMAT='(a)'
```

IDL will print something like the following:

```
sysalternates  
sysarticles  
syscolumns  
syspublications  
sysreferences  
systypes  
sysusers
```

```
mydata
```

Create a new table named “im_info” using SQL commands:

```
oDB->ExecutesQL, $
  "create table im_info (id integer, x integer," + $
  "y integer, data image, name char(50))"
```

Now create a Recordset object and connect to the table you just created:

```
oRS = obj_new('IDLdbRecordSet', oDB, table='im_info')
```

Add a record to the object. This record contains four fields that describe an image: the width of the image, the height of the image, the image data itself, and the name of the image.

```
oRS->AddRecord, 1, 400, 400, BYTSCl(DIST(400)), 'first image'
```

Move the current location in the table (the cursor position) to the first row:

```
status = oRS->MoveCursor(/FIRST)
```

You can check the value of the variable `status` and report on whether the move was successful:

```
IF(status NE 1) THEN BEGIN
  PRINT, 'Error moving database cursor'
  RETURN
ENDIF
```

Retrieve the information from this record into IDL variables:

```
X = oRS->GetField(1)           X size of image
Y = oRS->GetField(2)           Y size of image
image = oRS->GetField(3)       Image data
name = oRS->getField(4)        Image name
```

Create an IDL window to display the image:

```
WINDOW, COLORS=-5, TITLE=name, XSIZE=x, YSIZE=y
```

Reform the image into two dimensions (ODBC data is stored as a one-dimensional stream of bytes):

```
image = REFORM(image, 400, 400)
```

Display the image:

```
TVSCL, image
```

Now, delete the `im_info` table and destroy the database objects:

```
oDB->ExecutesQL, 'drop table im_info'
OBJ_DESTROY, oDB
```

ODBC SQL Syntax Notes

While this manual does not attempt to describe SQL syntax, the questions surrounding the following special ODBC syntax arise frequently enough to bear mentioning here. Consult your ODBC reference for detailed information on these topics.

Date, Time, and Timestamp Data

Because there are a wide variety of date and time formats in use by different databases, ODBC uses a special clause in the SQL statement to identify dates and times. The syntax is:

Syntax	Format
{d 'value'}	yyyy-mm-dd
{t 'value'}	hh:mm:ss
{ts 'value'}	yyyy-mm-dd hh:mm:ss

Table 2-1: Date, Time, and Timestamp Syntax

For example, to use a date-and-time timestamp, the SQL statement might look something like:

```
select time from events where time > { ts '1997-01-16 08:50:43' }
```

Scalar Functions

Scalar functions—string length, absolute value, or date, for example—require a special clause. To call a scalar function when selecting a result set, use syntax like:

```
{fn scalar-function}
```

where scalar-function is the name of the scalar function you are calling. For example, calling the UCASE function on a field might look something like this:

```
SELECT { fn UCASE(NAME) } FROM employee
```

Converting Data

ODBC provides a scalar function that requests that the data source convert data from one SQL data type to another. The syntax is:

```
{ fn CONVERT(value_expression, data_type) }
```

where *value_expression* is the name of a column from a table, a literal value, or the result of another scalar function, and *data_type* is one of ODBC's defined data types.

LIKE Predicate Escape Characters

When using an SQL LIKE predicate, the percent character (%) and the underscore character (_) have special meanings. You can include these characters as literals in a LIKE predicate by using an escape clause, which has the following syntax:

```
{ escape 'escape-character' }
```

where *escape-character* is a character used in front of the special character to force evaluation with its literal value.

For example, since the percent character matches zero or more of any character when used in a LIKE predicate, the string '%AAA%' would match any number any character, followed by three "A"s, followed by any number of any character. Using an escape clause in the LIKE predicate allows you to use the literal "%" in the string. For example:

```
select name where name like '\%AAA%' { escape '\' }
```

selects names that include the percent character, followed by three "A"s, followed by any number of any character. The backslash (\) is used to "escape" the percent character.

Outer Joins

ODBC supports the ANSI SQL-92 left outer join syntax. The syntax is:

```
{ oj outer-join }
```

where *outer-join* is:

```
table-reference LEFT OUTER JOIN  
{ table-reference | outer-join } ON search-condition
```

Consult your ODBC documentation for further details on outer joins.

Procedure Calls

An application can call a procedure in place of an SQL statement. The syntax for a procedure call is:

```
{ [?=] call procedure-name([parameter],[parameter],...) }
```

where *procedure-name* specifies the name of a procedure (stored on the data source) and *parameters* are parameters of the procedure.

Consult your ODBC documentation for further details on procedure calls.



Chapter 3:

IDL DataMiner API

This chapter describes the IDL DataMiner functions, objects, and methods.

How to Use this Chapter

The functions, object descriptions, and method descriptions for the IDL DataMiner are documented alphabetically in this chapter. The page or pages describing each class include references to sub- and super-classes, and to the methods associated with the class. Class methods are documented alphabetically following the description of the class itself.

A description of each method follows its name. Beneath the general description of the method are sections that describe the calling sequence for the method, its arguments (if any), and its keywords (if any). These sections are described below.

Calling Sequence

The “Calling Sequence” section shows the proper syntax for calling the method.

Procedure Methods

IDL procedures have the calling sequence:

```
PROCEDURE_NAME, Argument [, Optional_Arguments]
```

where PROCEDURE_NAME is the name of the procedure, *Argument* is a required parameter, and *Optional_Argument* is an optional parameter to the procedure.

IDL procedure methods have the calling sequence:

```
Obj→PROCEDURE_NAME, Argument [, Optional_Arguments]
```

where *Obj* is a valid object reference, PROCEDURE_NAME is the name of the procedure method, *Argument* is a required parameter, and *Optional_Argument* is an optional parameter to the procedure method.

Note

The square brackets around optional arguments are not used in the actual call to the procedure; they are simply used to denote the optional nature of the arguments within this document.

Functions

IDL functions have the calling sequence:

```
Result = FUNCTION_NAME(Argument [, Optional_Arguments])
```

where *Result* is the returned value of the function, *FUNCTION_NAME* is the name of the function, *Argument* is a required parameter, and *Optional_Argument* is an optional parameter.

IDL function methods have the calling sequence:

```
Result = Obj→FUNCTION_NAME(Argument [, Optional_Arguments])
```

where *Obj* is a valid object reference, *Result* is the returned value of the function method, *FUNCTION_NAME* is the name of the function method, *Argument* is a required parameter, and *Optional_Argument* is an optional parameter.

Note

The square brackets around optional arguments are not used in the actual call to the function; they are simply used to denote the optional nature of the arguments within this document. Note also that all arguments and keyword arguments to functions should be supplied *within* the parentheses that follow the function's name.

Arguments

The “Arguments” section describes each valid argument to the routine. Note that these arguments are positional parameters that must be supplied in the order indicated by the method's calling sequence.

Named Variables

Often, arguments that contain values upon return from the function or procedure (“output arguments”) are described as accepting “named variables”. A named variable is simply a valid IDL variable name. This variable *does not* need to be defined before being used as an output argument. Note, however that when an argument calls for a named variable, only a named variable can be used—sending an expression causes an error.

Keywords

The “Keywords” section describes each valid keyword argument to the routine. Note that keyword arguments are formal parameters that can be supplied in any order.

Keyword arguments are supplied to IDL methods by including the keyword name followed by an equal sign (“=”) and the value to which the keyword should be set. Note that keywords can be abbreviated to their shortest unique length. For example, the XSTYLE keyword can be abbreviated to XST.

Setting Keywords

When the documentation for a keyword says something similar to, “Set this keyword to enable logarithmic plotting,” the keyword is simply a switch that turns an option on and off. Usually, setting such keywords equal to 1 causes the option to be turned on. Explicitly setting the keyword to zero (or not including the keyword) turns the option off.

There is a “shortcut” that can be used to set a keyword equal to 1 without the usual syntax (i.e., `KEYWORD=1`). To “set” a keyword, simply preface it with a slash character (“/”). For example, to plot a wire mesh surface with a skirt around it, set the `SKIRT` keyword to the `SURFACE` routine as follows:

```
SURFACE, DIST(10), /SKIRT
```

Creating Database Objects

To create a database object, use the `OBJ_NEW` function (see “`OBJ_NEW`” in the *IDL Reference Guide*). The `Init` method for each class describes the arguments and keywords available when you are creating a new graphics object.

For example, to create a new database object, use the following call to `OBJ_NEW`:

```
myDB = OBJ_NEW('IDLdbDatabase')
```

Destroying Database Objects

To destroy a database object, use the `OBJ_DESTROY` function (see “`OBJ_DESTROY`” in the *IDL Reference Guide*). The `Cleanup` method is called to perform any class-specific cleanup operations before the object is destroyed.

For example, to remove database object, use the following call to `OBJ_DESTROY`:

```
OBJ_DESTROY, myDB
```

DIALOG_DBCONNECT()

Use the DIALOG_DBCONNECT function to connect to the DBMS via the standard ODBC dialog boxes. You will be prompted for information required to connect to the desired database. The function returns true (1) unless you selected the dialog's **Cancel** button, in which case it returns false (0).

Note

Due to Motif library inconsistencies, this dialog may fail on HP-UX and IBM AIX systems.

Calling Sequence

```
status = DIALOG_DBCONNECT(DBobj)
```

Arguments

DBobj

A valid Database object that is not already connected to a data source.

Keywords

DATASOURCE

Set this keyword equal to the name of a data source to which you wish to connect. (If you do not know the data source name in advance, you can use the GetDatasources method of the IDLdbDatabase object to retrieve a list of available data sources.)

USER_ID

Set this keyword equal to the user login name or ID used to log into the datasource.

PASSWORD

Set this keyword equal to the password corresponding to the user ID.

DIALOG_PARENT

Set this keyword equal to the widget ID of a widget over which the dialog should be positioned.

DB_EXISTS()

Use the DB_EXISTS function to determine if the database functionality is available on a specific platform. DB_EXISTS returns true (1) if the platform in use supports ODBC *and* the user is licensed to use the IDL DataMiner, or false (0) if it is not available.

Calling Sequence

```
status = DB_EXISTS()
```

IDLdbDatabase

An IDLdbDatabase object represents a connection to a datasource. Use the IDLdbDatabase object's instance data and methods to connect to, disconnect from, and perform operations to a Database Management System (DBMS).

Creation

There is no Init method for the IDLdbDatabase object. Use the following IDL command to create a new database object:

```
DBObj = OBJ_NEW('IDLdbDatabase')
```

Note that the returned database object is not considered valid until a connection to the datasource is made, either via the [IDLdbDatabase::Connect](#) method or the [DIALOG_DBCONNECT\(\)](#) function.

Destruction

Use the OBJ_DESTROY procedure to destroy a database object:

```
OBJ_DESTROY, DBObj
```

Any recordset objects associated with the database object will be destroyed along with the database object.

Methods

- [“IDLdbDatabase::Connect”](#) on page 48
- [“IDLdbDatabase::ExecuteSQL”](#) on page 49
- [“IDLdbDatabase::GetDatabases”](#) on page 50
- [“IDLdbDatabase::GetProperty”](#) on page 51
- [“IDLdbDatabase::GetTables”](#) on page 53
- [“IDLdbDatabase::SetProperty”](#) on page 54

IDLdbDatabase::Connect

Use the IDLdbDatabase::Connect procedure method to connect to the data source associated with a database object.

Calling Sequence

DBobj →[IDLdbDatabase::]Connect

Arguments

None.

Keywords

CONNECTION

Set this keyword equal to a raw ODBC connection string. No preprocessing is performed on the string before it is passed to the ODBC system. If this keyword is set, all other keywords are ignored. This keyword is useful mainly for advanced ODBC users.

DATASOURCE

Set this keyword equal to a string containing the name of a datasource to which you wish to connect. This name is dependent on the data source. A default data source can be specified in the ODBC initialization file. See [Chapter 4, “Understanding the ODBC.INI File”](#) for details.

USER_ID

Set this keyword equal to a string containing the user login name or ID used to log into the datasource.

PASSWORD

Set this keyword equal to a string containing the password corresponding to the user ID.

IDLdbDatabase::ExecuteSQL

Use the IDLdbDatabase::ExecuteSQL procedure method to execute an SQL statement. No results are expected from this statement; any that are received are discarded. You can use this method to perform actions such as creating or deleting a table. To use this method, the object must already be connected to a datasource.

Note

See “[ODBC SQL Syntax Notes](#)” on page 37 for information on some common questions about ODBC SQL syntax.

Calling Sequence

DBobj→ExecuteSQL, *strSQL*

Arguments

strSQL

A string that contains a valid SQL statement. This statement is executed in the data source referenced by the database object.

IDLdbDatabase::GetDatasources

The IDLdbDatabase::GetDatasources function method returns an array of available datasources. You do not need to make a connection before calling this method.

Note

Not all drivers support the ability to return a list of available data sources.

The GetDatasources method returns an array of IDL structures with the following two fields:

- DATASOURCE: The name of the database driver
- DESCRIPTION: A description of the driver.

Calling Sequence

```
Datasources = DBObj→GetDatasources()
```

Arguments

None.

IDLdbDatabase::GetProperty

Use the IDLdbDatabase::GetProperty procedure method to retrieve properties of the database object. You must have made a connection to a database before using this method.

Calling Sequence

DBobj→GetProperty

Arguments

None.

Keywords

CAN_GET_TABLES

Set this keyword equal to a named variable that will contain 1 (one) if the GetTables method is available for the current driver, or 0 (zero) otherwise.

DBMS_NAME

Set this keyword equal to a named variable that will contain the name of the Database that the object is associated with.

DRIVER_ODBC_LEVEL

Set this keyword equal to a named variable that will contain the ODBC level supported by the driver being used to connect to the database.

DBMS_VERSION

Set this keyword equal to a named variable that will contain the version number of the Database that the object is associated with.

DRIVER_VERSION

Set this keyword equal to a named variable that will contain the version number of the ODBC driver being used to connect to the database.

IS_CONNECTED

Set this keyword equal to a named variable that will contain 1 (one) if a connection to a database exists, or 0 (zero) otherwise.

IS_READONLY

Set this keyword equal to a named variable that will contain a 1 (one) if the database is read-only, or 0 (zero) if it is writable.

MAX_CONNECTIONS

Set this keyword equal to a named variable that will contain the maximum number of connections supported by the ODBC driver. If the maximum value is unknown, 0 is returned.

MAX_RECORDSETS

Set this keyword equal to a named variable that will contain the maximum number of recordsets supported by the ODBC driver. If the maximum is unknown, 0 is returned.

ODBC_LEVEL

Set this keyword equal to a named variable that will contain the ODBC level of the driver manager.

SQL_LEVEL

Set this keyword equal to a named variable that will contain the SQL level supported by the connection.

SQL_SERVER_NAME

Set this keyword equal to a named variable that will contain the SQL server name for this database connection.

USE_CURSOR_LIB

Set this keyword equal to a named variable that will contain 1 (one) if the USE_CURSOR_LIB property was set (via the SetProperty method), or 0 (zero) otherwise. Note that this keyword will return 1 if the USE_CURSOR_LIB property was set, *even if the cursor library was not subsequently loaded*. See [“IDLdbDatabase::SetProperty”](#) on page 54 for details.

USER_NAME

Set this keyword equal to a named variable that will contain the user name used during the connection to the datasource.

IDLdbDatabase::GetTables

The IDLdbDatabase::GetTables function method returns a list of available tables in the datasource. A connection is required before this method is called. This method is not supported for all drivers.

The GetTables method returns an array of IDL structures with the following fields:

- **QUALIFIER:** The table qualifier.
- **OWNER:** The owner of the table.
- **NAME:** The name of the table
- **TYPE:** The type of the table.

This function is not available in all ODBC drivers. Use the `CAN_GET_TABLES` keyword to the `GetProperty` method to determine whether this feature is available.

Calling Sequence

```
Tables = DBObj→GetTables()
```

Arguments

None.

IDLdbDatabase:: SetProperty

Use the IDLdbDatabase::SetProperty procedure method to set properties of the database object.

Calling Sequence

DBobj→SetProperty

Arguments

None.

Keywords

USE_CURSOR_LIB

Set this property to use the ODBC cursor library. The ODBC cursor library is used to emulate advanced functionality on data sources that don't support the advanced functions. If you find that advanced functionality is not available using your database's standard driver, try using the ODBC cursor library. Advanced functionality supported by the cursor library includes positioned updates, positioned deletes, and multi-directional cursor movement.

Note

This property must be set (or unset) before the connection to the data source is made. Once the connection is made, this property cannot be changed. The default is to not use the cursor library.

Warning

To support the above-mentioned operations, the cursor library constructs SQL search statements to locate the desired record. If the WHERE clause of the generated SQL statement selects more than one row, the operation will affect more than one record.

Warning

On some systems the ODBC cursor library is loaded dynamically. The ODBC system cannot detect whether the library was loaded successfully. Use this property with care.

VERBOSE

Set this keyword to enable verbose error messages. Normal error messages contain a text explanation (normally from the ODBC system) of the error. Verbose message include the following additional information:

- The name of the ODBC function that failed,
- The error code from the ODBC system,
- The error code from the database.

IDLdbRecordset

The IDLdbRecordset object contains a database table or the results from an SQL query.

Creation

To create a recordset object, a valid database object is required. Use the following IDL command to create a new recordset object:

```
RSObj = OBJ_NEW('IDLdbRecordset', DBobj, KEYWORD)
```

where *DBobj* is the object reference of the database object and *KEYWORD* is either SQL or TABLE:

N_BUFFERS

Set this keyword equal to the number of buffers to use when reading from the database. When a request is made to the database, records are read until the allocated buffers are filled; setting this number appropriately can greatly increase the performance of the IDL DataMiner. The default value is 10.

Note that increasing the number of buffers allocated increases the amount of memory used by the recordset object. You may need to experiment with other values to find the most efficient setting for your application.

SQL

A string that contains a valid SQL statement that selects records from the database.

TABLE

A string that contains the name of a table in the database. This table must be contained in the database referred to by *DBobj*.

Destruction

Use the OBJ_DESTROY procedure to destroy a recordset object:

```
OBJ_DESTROY, RSObj
```

Recordset objects are automatically destroyed if the database object they belong to is destroyed.

Methods

- “[IDLdbRecordset::AddRecord](#)” on page 58
- “[IDLdbRecordset::CurrentRecord](#)” on page 59
- “[IDLdbRecordset::DeleteRecord](#)” on page 60
- “[IDLdbRecordset::GetField](#)” on page 61
- “[IDLdbRecordset::GetProperty](#)” on page 62
- “[IDLdbRecordset::GetRecord](#)” on page 65
- “[IDLdbRecordset::MoveCursor](#)” on page 66
- “[IDLdbRecordset::NFields](#)” on page 67
- “[IDLdbRecordset::SetField](#)” on page 68

IDLdbRecordset::AddRecord

Use the IDLdbRecordset::AddRecord procedure method to add a record to a recordset. If you don't have permission to modify the recordset, an error is returned. The location in the recordset of the new record is dependent on the ODBC Driver, but in most cases it is added to the end of the recordset.

Calling Sequence

RSObj→AddRecord, [*field1*][*field2*]...[*fieldn*]

Arguments

Any arguments passed to this routine are used to initialize the new record. If these initialization parameters are not provided, the field is initialized to null. If the field cannot be set to null, it is initialized to 0.

Keywords

SET_AUTOINCREMENT

Normally when adding a record to the recordset the DataMiner skips setting the values on autoincrement fields. By setting this keyword, the DataMiner will attempt to set the value of the autoincrement field with the value provided. Note that the results from setting an autoincrement field is datasource dependent and might result in an error.

IDLdbRecordset::CurrentRecord

The IDLdbRecordset::CurrentRecord function method requests the current record number in a recordset. This method is driver-dependent. If the record number of the current record cannot be determined by the ODBC driver, this function returns a negative number.

Note

Because this function is driver-dependent, moving the cursor to the last record in a recordset will not necessarily allow you to determine the number of records in the recordset.

Calling Sequence

```
number = RSobj→CurrentRecord()
```

Arguments

None.

IDLdbRecordset::DeleteRecord

Use the `IDLdbRecordset::DeleteRecord` procedure method to delete the current record from a recordset. Any attempt to access this record after it has been deleted can result in an error. This method will fail if the SQL driver doesn't support positioned deletions to the recordset.

Calling Sequence

RSobj→DeleteRecord

Arguments

None.

IDLdbRecordset::GetField

Use the `IDLdbRecordset::GetField` function method to get the value of a field from the current record in the recordset. If the value of the field is NULL (as defined by SQL) a null value (zero or an empty string) is returned.

Calling Sequence

```
value = RSobj→GetField(iFieldName)
```

Arguments

iFieldName

The number of the field for which the value will be returned. Field numbers have a range of $0 \leq n < \text{number of fields}$.

Keywords

IS_NULL

Set this keyword equal to a named variable that will contain 1 (one) if the ODBC system returns a NULL value, or 0 (zero) if the value is non-NULL.

NULL_VALUE

Set this keyword equal to the value that should be returned by the `GetField` method if the value of the requested field is considered to be NULL by the ODBC system.

IDLdbRecordset::GetProperty

Use the IDLdbRecordset::GetProperty procedure method to get properties of the recordset.

Calling Sequence

RSobj→GetProperty

Arguments

None.

Keywords

CAN_MOVE_ABSOLUTE

Set this keyword equal to a named variable that will contain 1 (one) if the cursor for the recordset can move to an absolute record number.

CAN_MOVE_FIRST

Set this keyword equal to a named variable that will contain 1 (one) if the cursor for the recordset can move to the first record.

CAN_MOVE_LAST

Set this keyword equal to a named variable that will contain 1 (one) if the cursor for the recordset can move to the last record.

CAN_MOVE_NEXT

Set this keyword equal to a named variable that will contain 1 (one) if the cursor for the recordset can move to the next record.

CAN_MOVE_PRIOR

Set this keyword equal to a named variable that will contain 1 (one) if the cursor for the recordset can move to the previous record.

CAN_MOVE_RELATIVE

Set this keyword equal to a named variable that will contain 1 (one) if the cursor for the recordset can move to a record number relative to the current record number.

FIELD_INFO

Set this keyword equal to a named variable that will contain an array of field informational structures, one for each field in the result set. Field information is only available if the current recordset was generated from a table (that is, if the TABLE keyword was set when creating the recordset object). Information structures have the following fields (see the ODBC Manual for more information):

- TABLE_QUALIFIER: The table qualifier.
- TABLE_OWNER: The name of the table owner.
- TABLE_NAME: The name of the table.
- FIELD_NAME: The name of the field.
- TYPE_NAME: The datasource type name.
- PRECISION: Precision of the field.
- LENGTH: Length in bytes of the data.
- SCALE: The scale of the field.
- IS_NULLABLE: The field can contain null values.
- IS_AUTOINCREMENT: The field is an autoincrement field.
- IS_CASE_SENSITIVE: The value of the field is case sensitive.
- IS_UPDATABLE: The field can be updated.
- IDL_TYPE: The IDL type to which the field is mapped.

If a field is returned empty, this indicates that the driver doesn't support the query for that particular information.

GET_DATABASE

Set this keyword equal to a named variable that will contain an object reference to the database object used when the current recordset object was created.

IS_READONLY

Set this keyword equal to a named variable that will contain a 1 (one) if the database is read-only, or 0 (zero) if it is writable.

N_BUFFERS

Set this keyword equal to a named variable that will contain the number of buffers allocated for the recordset.

RECORDSET_SOURCE

Set this keyword equal to a named variable that will contain either the table name or SQL statement used to create the recordset.

IDLdbRecordset::GetRecord

Use the IDLdbRecordset::GetRecord function method to retrieve the value of the current record in an IDL anonymous structure. The field names of the structure are the field names of the recordset.

Note

Any *blob* data is placed in an IDL pointer and as such must be freed using the IDL PTR_FREE routine.

Calling Sequence

```
Result = RSObj->GetRecord()
```

Arguments

None.

Keywords

None.

Example

The following code fragment creates a database object and connects to the database, creates a recordset object, then moves to the first record in the recordset, retrieves the value of the record, and uses the IDL HELP procedure to display information on the record.

```
oDB = OBJ_NEW('IDLdbDatabase')
status = DIALOG_DBCONNECT(oDB)
oRS = OBJ_NEW('IDLdbRecordset', oDB, TABLE='table')
IF(oRS->MoveCursor(/FIRST) EQ 1) THEN BEGIN
    record = oRS->GetRecord()
    HELP, record, /STRUCTURE
ENDIF
```

IDLdbRecordset::MoveCursor

Use the IDLdbRecordset::MoveCursor function method to move the cursor in a given recordset. The function returns true (1) if the move operation was successful, or false (0) otherwise.

Calling Sequence

Result = *RSObj*→MoveCursor()

Arguments

None.

Keywords

ABSOLUTE

Set this keyword equal to the record number that the cursor should be moved to.

FIRST

Set this keyword to move the cursor to the first record in the recordset.

LAST

Set this keyword to move the cursor to the last record in the recordset.

NEXT

Set this keyword to move the cursor to the next record in the recordset.

PRIOR

Set this keyword to move the cursor to the previous record in the recordset.

RELATIVE

Set this keyword equal to the relative number of records that the cursor should be moved from its current position.

IDLdbRecordset::NFields

The IDLdbRecordset::NFields function method returns the number of fields in the recordset.

Calling Sequence

```
status = RSobj→NFields()
```

Arguments

None.

IDLdbRecordset::SetField

Use the IDLdbRecordset::SetField procedure method to set the value of a field in the current record of a recordset.

Calling Sequence

RSobj→SetField, *iFieldNumber*, *Value*

Arguments

iFieldNumber

The number of the field whose value is returned. Field numbers have a range of $0 \leq n < \text{number of fields}$.

Value

The value to which the field should be set. If the provided value is not of the correct type, it is converted.

Keywords

NULL

Set this keyword to set the value of the field to NULL. Null is a special value used in database systems to indicate that a specific field does not contain a valid value.



Chapter 4:

Understanding the ODBC.INI File

The following topics are discussed in this chapter:

Overview	70	ODBC.INI File Example	74
ODBC.INI File Format	71		

Overview

The ODBC . INI is an initialization file used by the ODBC Driver Manager and ODBC drivers. Although this file has a slightly different name depending upon what platform you are using, the format and information contained in the file is the same. For any general discussion, the file name appears in uppercase letters.

- **WINDOWS:** For Windows users, ODBC . INI is a text file called ODBC . INI. The file is located in your WINDOWS directory. Although the ODBC . INI file is described in the following sections, Windows users *should not* modify this file—the ODBC Administrator program modifies it for you. The following sections are intended for informational purposes only.

Warning

Windows users should never modify the ODBC . INI file directly. The contents of this file are changed based on the data source set up and modifications you make using the ODBC Administrator. Modifying the ODBC . INI file directly may result in data source configuration or connection errors.

- **UNIX:** For UNIX users, ODBC . INI is a text file called . odbc . ini. The file is located in your home directory. Initially, a template file called odbc . ini resides in the odbc root directory of the IDL Distribution where the ODBC software was installed. Before using an ODBC driver, each user must copy this file to their home directory and rename it . odbc . ini. (The dot at the beginning of the name follows UNIX conventions for application initialization files.) UNIX users are responsible for modifying their . odbc . ini file using a text editor. For driver-specific . odbc . ini changes and information, refer to the appropriate driver chapter in this manual.

ODBC.INI File Format

The ODBC.INI file is made up of the following sections:

- **ODBC Data Sources.** This section lists the name of each data source and describes its associated driver.
- **Data Source Specification.** For each data source listed in the ODBC Data Sources section, there is a section that contains additional information about that data source.
- **Default Data Source Specification.** This section is optional and specifies the default data source to use when no data source is specified.
- **ODBC Options.** This section specifies the ODBC root directory and the ODBC options that may be enabled or disabled.

ODBC Data Sources

Each entry in the ODBC Data Sources section lists a data source and a description of the driver it uses. Entries in this section have the following format:

```
data_source_name = driver_description
```

The `data_source_name` identifies the data source to which the driver connects. You choose this name. This field is required.

The `driver_description` describes the driver to which the data source connects. This field is optional.

For example, to define an Agencies data source that uses the SYBASE SQL Server 10 driver, the ODBC.INI entry would look like the following:

```
[ODBC Data Sources]
Agencies=Sybase SQL Server 10
```

Data Source Specification

Each data source listed in the ODBC Data Sources section has its own data source specification section. This section has the following format:

```
[Data_source_name]
Driver=path_specification
Attribute=keyword_value
```

The `data_source_name` is the name defined in the ODBC Data Sources section of the ODBC.INI file.

The `path_specification` is the full path name to the dynamic link library (Windows), or the full path to the driver shared library (UNIX).

Each Attribute and `keyword_value` pair specifies the value of a driver-specific keyword. Each driver has its own set of keywords. For driver-specific keywords and attributes, refer to the *ODBC DriverSet Reference* chapter. There can be any number of Attribute/keyword pairs included in the Data Source Specification.

For example, the data source called Agencies connects to a Sybase SQL Server 10 driver for UNIX called `dmsyb13.so`. The database that Agencies accesses is also called agencies and it resides on the SYBASE10 server. The data source specification entry for the Agencies data source would look like the following:

```
[Agencies]
Driver=/opt/odbc/drivers/dmsyb13.so
Server=SYBASE10
Database=agencies
UID=marvin
```

In this example, the driver-specific keywords for the Sybase driver are Server, Database, and UID.

Default Data Source Specification

This section is optional. The Default Data Source specification contains information about the default data source. This data source is called Default and has the same format as any other data source specification section. However, the Default data source is not listed in the ODBC Data Sources section.

The following example shows a Default data source specification entry for an Oracle7 database.

```
[Default]
Driver=/opt/odbc/drivers/dmor713.so
Server=t:mickey:customers
UID=marvin
```

In this example, the driver-specific keywords for the Oracle7 driver are Server and UID. The Server keyword identifies the SQL*Net connect string for the ORACLE7 server called customers.

ODBC Options

The ODBC Options section specifies the ODBC root directory (UNIX only) and indicates whether tracing is enabled or disabled. With tracing, all ODBC function calls made from an application can be logged to the specified trace file.

Warning

For UNIX users: This section of the `.odbc.ini` file is recommended for UNIX installations so that the Driver Manager can find the message files. The Driver Manager also uses this section to load the Cursor Library and the Connection Dialog Library. At a minimum, the [ODBC] section must contain the `InstallDir` keyword with the value set to the path in which the DriverSet is installed.

This section has the following format:

```
InstallDir=odbc_path
Trace= 1 or 0
TraceFile=log_path
TraceDll=odbc_path/odbctrac.so
```

The `odbc_path` is the full path to the ODBC root directory. This option appears only in the UNIX version of the `ODBC.INI` file.

If the `TRACE` keyword is set to 0, tracing is disabled. If the `TRACE` keyword is set to 1, tracing is enabled.

The `log_path` is the full path to the specified trace file that is logging the ODBC function calls. If a trace file is not specified and tracing is enabled, logging information is written to the `sql.log` file located in your current directory.

The `TraceDll` keyword indicates the shared library that contains the ODBC tracking system.

```
[ODBC]
InstallDir=/opt/odbc
Trace=1
TraceFile=/opt/odbc/drivers/trace.log
TraceDll=/opt/odbc/lib/odbctrac.so
```

ODBC.INI File Example

UNIX: The following example shows a UNIX `.odbc.ini` file.

```
[ODBC Data Sources]
Informix9=INTERSOLV 3.11 Informix 9 Driver
Text=INTERSOLV 3.11 Text Driver

[Text]
Driver=/opt/odbc/lib/dmtxt13.so
Description=Text driver
Database=/home/kirk/dmtest
AllowUpdateAndDelete=1

[Informix9]
Driver=/opt/lib/dminf913.so
Description=Informix9
Database=odbc
HostName=informixhost
LogonID=odbc01
Password=odbc01

[ODBC]
InstallDir=/opt/odbc
Trace=1
TraceFile=/opt/odbc/drivers/trace.log
TraceDll=/opt/odbc/lib/odbctrac.so
```



Chapter 5: Using Intersolv ODBC Drivers

The following topics are discussed in this chapter:

Supported Drivers	76	Connect ODBC for Text	127
Connect ODBC for INFORMIX	79	The UNIX Environment	149
Connect ODBC for Oracle	93	Locking and Isolation Levels	152
Connect ODBC for Sybase	109		

Supported Drivers

The following table describes the drivers that are included on your distribution CD-ROM:

Supported Databases	Driver Name	Supported Platforms
INFORMIX 5.x, 6.x, or 7.x	INFORMIX	Windows 95/NT 4.0/NT 4.0 for DEC Alpha Sun Solaris 2.6 (requires Informix Connect 7.23) HP-UX 10.20 (requires Informix Connect 7.23) IRIX 6.4
INFORMIX 7.x or 9.x	INFORMIX 9	Windows 95/NT 4.0 Sun Solaris 2.6 (requires Informix Connect 9.13) AIX 4.3 (requires Informix Connect 9.14) IRIX 6.4
Oracle 7.x	Oracle7	Windows 95/NT 4.0/NT 4.0 for DEC Alpha Sun Solaris 2.6 AIX 4.3 HP-UX 10.20 MacOS 8.1 (requires SQL*Net 2.x)

Table 5-1: Supported ODBC Drivers for DataMiner

Supported Databases	Driver Name	Supported Platforms
Oracle 8.0	Oracle 8	Windows 95/NT 4.0/NT 4.0 for DEC Alpha (requires Oracle's Net8 Client version 8.04 or higher) Sun Solaris 2.6 (requires Oracle's Net8 Client 8.0.3 or higher) AIX 4.3 (requires Oracle's Net8 Client 8.0.3 or higher) IRIX 6.4 (requires Oracle N32 Client Development Kit, Version 8.0.5.0.0 (Oracle Part Number: Z24604-02) or later) MacOS 8.1 (SQL*Net 2.x)
SQL Server 4.9.2, SQL Server System 10, System 11, and Adaptive Server 11.5 and 11.9	Sybase	Windows 95/NT 4.0/NT 4.0 for DEC Alpha Sun Solaris 2.6 (System 10 and 11 only) AIX 4.3 (System 11 only) HP-UX 10.20 (System 10, System 11, and Adaptive Server 11.5 and 11.9) IRIX 6.4 MacOS 8.1 (System 10 and 11 only)
MS SQL Server 6.5	SQL Server	Windows 95/NT 4.0
ASCII text files	Text	Windows 95/NT 4.0 Sun Solaris 2.6 AIX 4.3 HP-UX 10.20 MacOS 8.1

Table 5-1: Supported ODBC Drivers for DataMiner

In addition, ODBC compliant Drivers can be obtained from other sources and can be used with the IDL DataMiner package. These can be obtained directly from database vendors and other third-party software providers. Most notably, Microsoft provides

ODBC drivers for FoxPro, Access, and Excel. For more information, visit the Microsoft Web site at www.microsoft.com/odbc.

Connect ODBC for INFORMIX

Connect ODBC for INFORMIX supports two separate drivers. Connect ODBC for INFORMIX (the “INFORMIX driver”) supports multiple connections to the INFORMIX database system versions 5.x, 6.x, or 7.x in the Windows 9x, Windows NT, and UNIX environments.

Connect ODBC for INFORMIX 9 (the “INFORMIX 9 driver”) supports multiple connections to the INFORMIX database system versions 7.x and 9.x in the Windows 9x, Windows NT, and UNIX environments.

System Requirements

The following section lists requirements for all supported platforms.

Windows 9x and Windows NT

Both INFORMIX and INFORMIX 9 are supported on Windows 9x and Windows NT.

INFORMIX

To access remote INFORMIX 5.x, 6.x, or 7.x databases through the INFORMIX driver, you need INFORMIX-Connect 7.2 for Windows 9x and Windows NT from INFORMIX.

Note

The DataDirect INFORMIX driver for Windows 9x and Windows NT does not work with versions of INFORMIX-Connect earlier than 7.2.

Use the SETNET32.EXE utility supplied with INFORMIX-Connect 7.2 to define servers and the location of the INFORMIX directory. Use ILOGIN.EXE to test your connection to the INFORMIX server.

The path to the ISQLT07C.DLL must be in your PATH environment variable. If it is not and you attempt to configure a data source, a message similar to the following appears:

```
The setup routines for the INTERSOLV 3.00 32-BIT INFORMIX ODBC
driver could not be loaded due to system error code 126.
```

When you click **OK**, the following message appears:

```
Could not load the setup or translator library.
```

INFORMIX 9

To access remote INFORMIX 7.x or 9 databases through the INFORMIX 9 driver, you need INFORMIX-Connect 9.1.3 or greater for Windows 9x and Windows NT from INFORMIX.

Use the `SETNET32.EXE` utility supplied with INFORMIX-Connect 9.1.3 to define servers and the location of the INFORMIX directory. Use `ILOGIN.EXE` to test your connection to the INFORMIX server.

The path to the `ISQLT09A.DLL` must be in your `PATH` environment variable. If it is not and you attempt to configure a data source, a message similar to the following appears:

```
The setup routines for the INTERSOLV 3.00 32-BIT INFORMIX ODBC
driver could not be loaded due to system error code 126.
```

When you click **OK**, the following message appears:

```
Could not load the setup or translator library.
```

UNIX (AIX, HP-UX, IRIX, and Solaris for SPARC)

The environment variable `INFORMIXDIR` must be set to the directory where you have installed the INFORMIX client.

For example, the following syntax is valid for C-shell users:

```
setenv INFORMIXDIR /databases/informix
```

For Bourne- or Korn-shell users, the following syntax is valid:

```
INFORMIXDIR=/databases/informix;export INFORMIXDIR
```

In addition, the `INFORMIXSERVER` variable must be set to the name of the INFORMIX server (as defined in your `$INFORMIXDIR/ext/sqlhosts` file). For further details, refer to the *INFORMIX Online Dynamic Server Administrator's Guide, Volume 2* or the *INFORMIX UNIX Installation Guide*.

INFORMIX

The INFORMIX driver under UNIX requires INFORMIX-Connect or ESQL-C 7.23.

INFORMIX 9

To access remote INFORMIX 7.x or 9 databases through the INFORMIX 9 driver, you need INFORMIX-Connect or ESQL-C 9.1.3 for Solaris and HP-UX, and INFORMIX-Connect 9.1.4 for AIX.

Configuring Data Sources

Note

In the UNIX environment, there is no ODBC Administrator. To configure a data source in the UNIX environment, you must edit the system information file using the attributes in [Table 5-2](#). You must also edit this file to perform a translation. For information about this file, see [“The UNIX Environment”](#) on page 149.

To configure an INFORMIX data source:

1. Start the ODBC Administrator to display a list of data sources.
2. If you are configuring an existing data source, select the data source name and click **Configure** to display the ODBC INFORMIX Driver Setup dialog box.

If you are configuring a new data source, click **Add** to display a list of installed drivers. Select the INFORMIX driver and click **Finish** to display the **ODBC INFORMIX Driver Setup** dialog box.

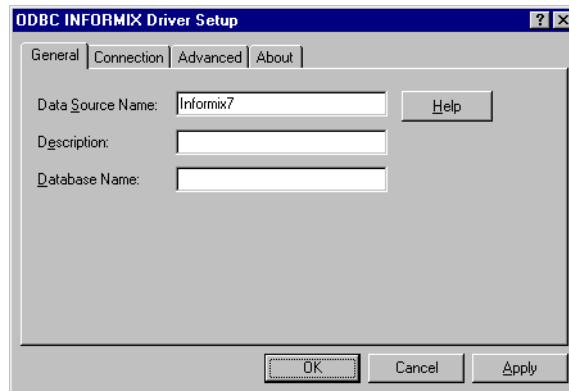


Figure 5-1: The **ODBC INFORMIX Driver Setup** dialog box

3. Specify values as follows; then, click **Apply**:

Data Source Name: A string that identifies this INFORMIX data source configuration in the system information. Examples include “Accounting” or “INFORMIX-Serv1.”

Description: An optional long description of a data source name. For example, “My Accounting Database” or “INFORMIX 7 files on Server number 1.”

Database Name: The name of the database to which you want to connect by default.

- Click the **Connection** tab to configure additional, optional settings for the data source.

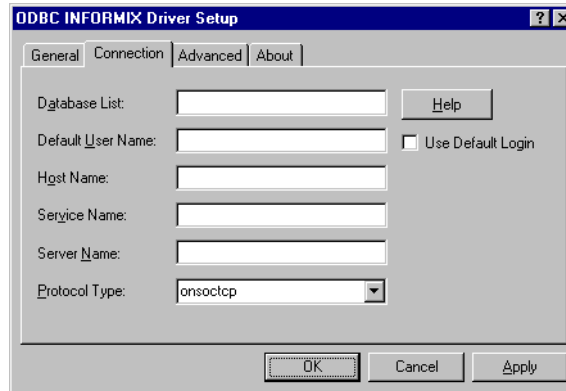


Figure 5-2: ODBC INFORMIX Driver Setup Connection tab

- Specify values as follows; then, click **Apply**.

Database List: The list of databases that will be displayed in the Logon dialog box if **Get DB List From Informix** on the **Advanced** tab is *not* checked.

Default User Name: The name of the user as specified on the INFORMIX server.

Use Default Login: Select this check box to read the Logon ID and Password entries directly from the INFORMIX registry. The check box is cleared by default; that is, logon information is read from the system information, the connection string, or the **Logon to INFORMIX** dialog box.

Host Name: The name of the machine on which the INFORMIX server resides.

Service Name: The name of the service as it appears on the host machine. This service is assigned by the system administrator. The name you specify is displayed in the **INFORMIX Server Options** dialog box.

Server Name: The name of the INFORMIX server as it appears in the sqlhosts file.

Protocol Type (Windows only): The protocol used to communicate with the server. Specify one or more values; separate the names with commas. Values can be `olsocspx`, `olsoctcp`, `onsocspx`, `onsoctcp`, `seipepip`, `sesocspx`, and/or `sesoectp`.

Click the **Advanced** tab to configure additional, optional settings for the data source.

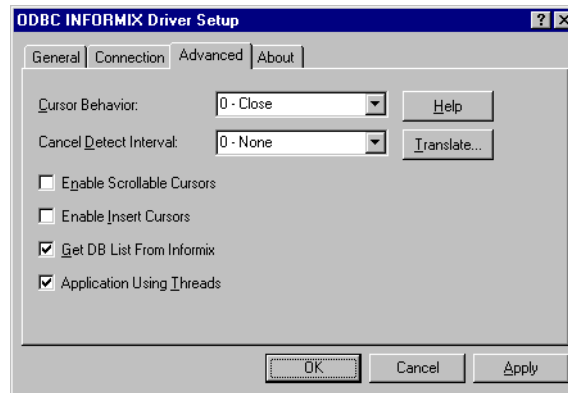


Figure 5-3: ODBC INFORMIX Driver Setup Advanced tab

6. Specify values as follows; then, click **Apply**:

Cursor Behavior: Holds cursor at the current position when the transaction ends if you select **Preserve**. Otherwise, leave this set to **Close**. Selecting **Preserve** may impact the performance of your database operations.

Cancel Detect Interval: Lets you cancel long-running queries in threaded applications. Select a value to determine how often the driver checks whether a request has been canceled using `SQLCancel`. For example, if `CDI=5`, then for every pending request, the driver checks every five seconds to see whether the user has canceled execution of the query using `SQLCancel`. The default is 0, which means that requests will not be canceled until the request has completed execution.

Note

IDL does not support multi-threading.

Enable Scrollable Cursors: Determines whether the driver provides scrollable cursors. The check box is cleared by default (no use of scrollable

cursors). The INFORMIX driver can use scrollable cursors only if there are no long columns (SQL_LONGVARCHAR or SQL_LONGVARBINARY) in a **Select** list. If you select this check box, you must not include long columns in the **Select** list.

Enable Insert Cursors: Determines whether the driver can use Insert cursors during parameterized inserts. Using Insert cursors improves performance during multiple Insert operations using the same statement. This option enables insert data to be buffered in memory before being written to disk. When this check box is cleared (the default), the driver does not use Insert cursors.

Get DB List From Informix: Determines whether the driver requests the database list to be returned from the INFORMIX server or from the database list that the user entered during driver setup.

When the check box is selected, the driver requests the database list from the INFORMIX server. When the check box is cleared, the driver uses the list that was entered by the user at driver setup.

Application Using Threads: A setting that ensures that the driver works with multi-threaded applications. You can clear this check box when using the driver with single-threaded applications. Clearing this check box avoids the additional processing required for ODBC thread-safety standards.

Note

IDL does not support multi-threading.

Translate: Click **Translate** to display the **Select Translator** dialog box, which lists the translators specified in the ODBC Translators section of the system information. INTERSOLV provides a translator named INTERSOLV OEM ANSI that translates your data from the IBM PC character set to the ANSI character set.

Select a translator; then, click **OK** to close this dialog box and perform the translation.

7. Click **OK** or **Cancel**. If you click **OK**, the values you have specified become the defaults when you connect to the data source. You can change these defaults by using this procedure to reconfigure your data source. You can override these defaults by connecting to the data source using a connection string with alternate values.

Connecting to a Data Source Using a Logon Dialog Box

Some ODBC applications display a logon dialog box when you are connecting to a data source. In these cases, the data source name has already been specified. For INFORMIX 5 or INFORMIX 7.2, the dialog box is as follows:

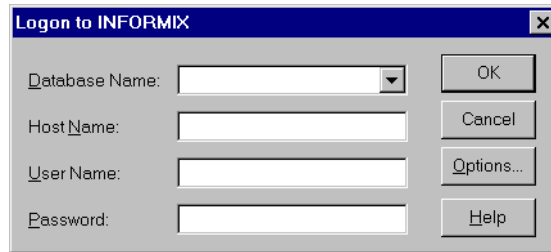


Figure 5-4: The **Logon to INFORMIX** dialog box

In this dialog box, do the following:

1. Type the name of the database you want to access or select the name from the **Database Name** drop-down list. The names on the list are determined by the status of the **Get DB List From Informix** checkbox on the **Advanced** tab. If the box is checked, the names displayed are from the user-entered list. If it is not checked, the names displayed are returned from the INFORMIX server.
2. Type the name of the server (host name) on which INFORMIX resides.
3. If required, type your user name as specified on the INFORMIX server.
4. If required, type your password.
5. Optionally, click **Options** to display the **INFORMIX Server Options** dialog box, where you can change the Service Name, Server Name, and Protocol

Type that you specified in the **ODBC INFORMIX Driver Setup** dialog box. Click **OK** to save your changes.

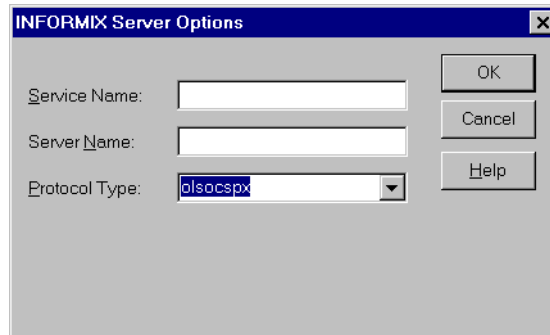


Figure 5-5: The **INFORMIX Server Options** dialog box

6. Click **OK** to complete the logon and to update these values in the system information.

Connecting to a Data Source Using a Connection String

If your application requires a connection string to connect to a data source, you must specify the data source name that tells the driver which section in the system information to use for the default connection information. Optionally, you may specify *attribute=value* pairs in the connection string to override the default values stored in the system information. These values are not written to the system information.

You can specify either long or short names in the connection string. The connection string has the form:

```
DSN=data_source_name[;attribute=value
[;attribute=value]...
```

An example of a connection string for INFORMIX is:

```
DSN=INFORMIX TABLES;DB=PAYROLL
```

The following table gives the long and short names for each attribute, as well as a description.

Note

To configure a data source in the UNIX environment, you must edit the system information file. This file accepts only long names for attributes. For information about this file, see [“The UNIX Environment”](#) on page 149.

The defaults listed in the table are initial defaults that apply when no value is specified in either the connection string or in the data source definition in the system information. If you specified a value for the attribute when configuring the data source, that value is your default.

Attribute	Description
ApplicationUsing Threads (AUT)	<p>ApplicationUsingThreads={0 1}. Ensures that the driver works with multi-threaded applications. The default is 1, which makes the driver thread-safe. When using the driver with single-threaded applications, you may set this option to 0 to avoid additional processing required for ODBC thread safety standards.</p> <p>Note - IDL does not support multi-threading.</p>
CancelDetect Interval (CDI)	<p>Lets you cancel long-running queries in threaded applications. Select a value to determine how often the driver checks whether a request has been canceled using SQLCancel. For example, if CDI=5, then for every pending request, the driver checks every five seconds to see whether the user has canceled execution of the query using SQLCancel. The default is 0, which means that requests will not be canceled until a request has completed execution.</p> <p>Note - IDL does not support multi-threading.</p>
CursorBehavior (CB)	<p>CursorBehavior={0 1}. This attribute determines whether cursors will be preserved or closed at the end of each transaction. The initial default is 0 (close). Set this attribute to 1 if you want cursors to be held at the current position when the transaction ends. The value CursorBehavior=1 may impact the performance of your database operations.</p>
Database (DB)	<p>The name of the database to which you want to connect.</p>

Table 5-2: : INFORMIX Connection String Attributes

Attribute	Description
DataSourceName (DSN)	A string that identifies an INFORMIX data source configuration in the system information. Examples include “Accounting” or “INFORMIX-Serv1.”
EnableInsert Cursors (EIC)	EnableInsertCursors={0 1}. Determines whether the driver can use Insert cursors during parametrized inserts. The initial default value is 1 (driver uses Insert cursors). Using Insert cursors improves performance during multiple Insert operations using the same statement. This option enables insert data to be buffered in memory before being written to disk. When EnableInsertCursors=0, the driver does not use Insert cursors.
EnableScrollable Cursors (ESC)	EnableScrollableCursors={0 1}. This attribute determines whether the driver provides scrollable cursors. The initial default value is 0 (no use of scrollable cursors). The INFORMIX driver can use scrollable cursors only if there are no long columns (SQL_LONGVARCHAR or SQL_LONGVARIABLE) in a Select list. If you set this option to use scrollable cursors (EnableScrollableCursors=1), you must not include long columns in the Select list.
GetDBListFrom Informix (GDBLFI)	GetDBListFromInformix={0 1}. This attribute determines whether the driver requests the database list to be returned from the INFORMIX server or from the database list that the user entered at driver setup. When set to 1, the initial default, the driver requests the database list from the INFORMIX server. When set to 0, it uses the list that was entered by the user at driver setup.
HostName (HOST)	The name of the machine on which the INFORMIX server resides.
LogonID (UID)	Your user name as specified on the INFORMIX server.
Password (PWD)	A password.

Table 5-2: : INFORMIX Connection String Attributes

Attribute	Description
Protocol (PRO) (Windows Only)	Protocol={olsocsp olsocetp onsocsp onsoctep seipcpip sesocsp sesocetp}. The protocol used to communicate with the server. You can specify one or more values; separate the names with commas.
ServerName (SRVR)	The name of the server running the INFORMIX database.
Service (SERV)	The name of the service as it appears on the host machine. This service is assigned by the system administrator.
UseDefaultLogin (UDL)	UseDefaultLogin={0 1}. Specify 1 to read the Logon ID and Password directly from the INFORMIX registry. The default is 0; that is, logon information is read from the system information, the connection string, or the Logon to INFORMIX dialog box.

Table 5-2: : INFORMIX Connection String Attributes

Data Types

The following table shows how the INFORMIX data types map to the standard ODBC data types.

INFORMIX	ODBC
Byte ¹	SQL_LONGVARIABLE
Char	SQL_CHAR
Date	SQL_TYPE_DATE
Datetime year to fraction(5)	SQL_TYPE_TIMESTAMP
Datetime year to fraction(f) ²	SQL_TYPE_TIMESTAMP
Datetime year to second	SQL_TYPE_TIMESTAMP
Datetime year to day	SQL_TYPE_DATE
¹ Not supported for Standard Engine Databases ² Fraction(f) types are mapped to fraction(5) in the driver. The precision is type dependent and the scale as 5.	

Table 5-3: : INFORMIX Data Types

INFORMIX	ODBC
Datetime hour to second	SQL_TYPE_TIME
Datetime hour to fraction(f) ²	SQL_TYPE_TIME
Decimal	SQL_DECIMAL
Float	SQL_DOUBLE
Integer	SQL_INTEGER
Interval year(p) to year	SQL_INTERVAL_YEAR
Interval year(p) to month	SQL_INTERVAL_YEAR_TO_MONTH
Interval month(p) to month	SQL_INTERVAL_MONTH
Interval day(p) to day	SQL_INTERVAL_DAY
Interval day(p) to hour	SQL_INTERVAL_DAY_TO_HOUR
Interval day(p) to minute	SQL_INTERVAL_DAY_TO_MINUTE
Interval day(p) to second	SQL_INTERVAL_DAY_TO_SECOND
Interval day(p) to fraction(f) ²	SQL_INTERVAL_DAY_TO_SECOND
Interval hour(p) to hour	SQL_INTERVAL_HOUR
Interval hour(p) to minute	SQL_INTERVAL_HOUR_TO_MINUTE
Interval hour(p) to second	SQL_INTERVAL_HOUR_TO_SECOND
Interval hour(p) to fraction(f) ²	SQL_INTERVAL_HOUR_TO_SECOND
Interval minute(p) to minute	SQL_INTERVAL_MINUTE
Interval minute(p) to second	SQL_INTERVAL_MINUTE_TO_SECOND
Interval minute(p) to fraction(f) ²	SQL_INTERVAL_MINUTE_TO_SECOND
Interval second(p) to second	SQL_INTERVAL_SECOND
Interval second(p) to fraction(f) ²	SQL_INTERVAL_SECOND
Interval fraction to fraction(f) ²	SQL_VARCHAR
¹ Not supported for Standard Engine Databases ² Fraction(f) types are mapped to fraction(5) in the driver. The precision is type dependent and the scale as 5.	

Table 5-3: : INFORMIX Data Types

INFORMIX	ODBC
Money	SQL_DECIMAL
Serial	SQL_INTEGER
Smallfloat	SQL_REAL
Smallint	SQL_SMALLINT
Text ¹	SQL_LONGVARCHAR
Varchar ¹	SQL_VARCHAR
¹ Not supported for Standard Engine Databases ² Fraction(f) types are mapped to fraction(5) in the driver. The precision is type dependent and the scale as 5.	

Table 5-3: : INFORMIX Data Types

INFORMIX 9

The following table shows how the INFORMIX 9 data types map to the standard ODBC data types. These types are in addition to the INFORMIX data types described in [Table 5-3](#).

INFORMIX 9	ODBC
Blob	SQL_LONGVARBINARY
Boolean	SQL_BIT
Clob	SQL_LONGVARCHAR
Int8	SQL_BIGINT
Lvarchar	SQL_VARCHAR
Serial8	SQL_BIGINT

Table 5-4: : INFORMIX 9 Data Types

The INFORMIX 9 driver does not support any complex data types (for example, set, multiset, list, and named/unnamed abstract types). When the driver encounters a complex type it will return an Unknown Data Type error (SQL State HY000).

Isolation and Lock Levels Supported

If connected to an Online Server, INFORMIX supports isolation levels 0 (read uncommitted), 1 (read committed), and 3 (serializable). The default is 1. The Standard Engine supports isolation level 0 (read uncommitted) only.

INFORMIX also supports an alternative isolation level 1, called cursor stability. Your ODBC application can use this isolation level by calling `SQLSetConnectAttr` (1040,1).

Additionally, if transaction logging has not been enabled for your database, then transactions are not supported by the driver (the driver is always in auto-commit mode).

INFORMIX supports page-level and row-level locking.

See “[Locking and Isolation Levels](#)” on page 152 for a discussion of these topics.

ODBC Conformance Level

The INFORMIX driver supports the functions listed in [Chapter 6, “ODBC API and Scalar Functions”](#). In addition, the following X/Open functions are supported:

- `SQLProcedures`
- `SQLColumnPrivileges`
- `SQLTablePrivileges`
- `SQLPrimaryKeys`
- `SQLForeignKeys`
- `SQLProcedureColumns`

The driver also supports scrollable cursors with `SQLExtendedFetch` or `SQLFetchScroll` if the connection attribute `EnableScrollableCursors` is set to 1. The driver supports the core SQL grammar.

Number of Connections and Statements Supported

The INFORMIX driver supports multiple connections and multiple statements per connection to the INFORMIX database system.

For more detail on how to use IDL DataMiner classes to perform actions on a DBMS, see [Chapter 3, “IDL DataMiner API”](#). For information on IDL commands and syntax, see the *IDL Reference Guide*.

Connect ODBC for Oracle

Connect ODBC for Oracle supports two separate drivers. Connect ODBC for Oracle (the “Oracle driver”) supports the Oracle 7 database system. The Oracle driver is supported in the Windows 9x, Windows NT, Macintosh Power PC, and UNIX environments.

Connect ODBC for Oracle 8 (the “Oracle 8 driver”) supports the Oracle 8 database system. The Oracle 8 driver is supported in the Windows 9x, Windows NT, Macintosh Power PC, and UNIX environments.

See the README file shipped with your INTERSOLV DataDirect product for the file names of the Oracle drivers.

System Requirements

The following section lists requirements for all supported platforms.

Windows 9x and Windows NT

Both Oracle and Oracle 8 client information for Windows 9x and Windows NT is listed below.

Oracle

The Oracle SQL*Net product is required to access remote Oracle databases. The appropriate DLLs for the current version of SQL*Net and OCIW32.DLL must be on your path. For example, SQL*Net 2.3 requires ORA73.DLL, CORE35.DLL, NLSRTL32.DLL, and CORE350.DLL, as well as OCIW32.DLL. If you attempt to configure an Oracle 7 data source and you do not have these DLLs on your path, a message similar to the following appears:

```
The setup routines for the INTERSOLV 3.00 32-BIT Oracle driver
could not be loaded due to system error code 126.
```

When you click **OK**, the following message appears:

```
Could not load the setup or translator library.
```

Oracle 8

The Oracle Net8 Client version 8.0.4, or greater, is required to access remote Oracle 8 databases. For Alpha NT systems, version 8.0.3 is required. On Intel systems, the appropriate DLLs for the Oracle Net8 Client must be on your path, for example, ORA804.DLL, PLS804.DLL, and OCI.DLL. If you attempt to configure an Oracle 8

data source and you do not have these DLLs on your path, a message similar to the following appears:

```
The setup routines for the INTERSOLV 3.00 32-BIT Oracle driver
could not be loaded due to system error code 126.
```

When you click **OK**, the following message appears:

```
Could not load the setup or translator library.
```

UNIX

Both Oracle and Oracle 8 client information for UNIX is listed below.

Oracle and Oracle 8

Before you can use the Oracle data source, you must have the Oracle SQL*Net or Net8 drivers you plan to use installed on your workstation in the \$ORACLE_HOME source tree. ORACLE_HOME is an environment variable created by the Oracle installation process that identifies the location of your Oracle client components.

Oracle refers to the runtime Oracle component as “Oracle RDBMS.” From the Oracle RDBMS product, the Oracle driver depends on the executables in \$ORACLE_HOME/bin and the interface libraries in \$ORACLE_HOME/rdbsms/lib.

Set the environment variable ORACLE_HOME to the directory where you installed the Oracle RDBMS, SQL*Net, or Net8 product. For example, for C-shell users, the following syntax is valid:

```
setenv ORACLE_HOME /databases/oracle
```

For Bourne- or Korn-shell users, the following syntax is valid:

```
ORACLE_HOME=/databases/oracle;export ORACLE_HOME
```

Note

For IRIX, you need to set the environment variable ORACLE_N32_HOME. Refer to your Oracle documentation for more information.

Building the Required Oracle 7 SQL*Net Shared Library

The Oracle driver requires a one-time site linking to build an Oracle 7 SQL*Net driver on AIX and, for Oracle 7.1 only, on Solaris and HP-UX. This site linking binds your unique Oracle 7 SQL*Net configuration into the file, which is used by the Oracle driver to access local and remote Oracle databases.

Before you build the Oracle 7 SQL*Net shared library, install Oracle and set the environment variable ORACLE_HOME to the directory where you installed Oracle.

Connect ODBC provides a script, `genclntsh`, that builds the Oracle 7 SQL*Net driver. This script is in the `scr/oracle` directory.

The following command builds the Oracle 7 SQL*Net shared library:

```
genclntsh
```

Building the Required Oracle Net8 Shared Library on Solaris

Under Oracle 8.0.3 or 8.0.4 for Solaris, the Oracle 8 driver requires a one-time site linking to build a replacement Oracle Net8 driver. This site linking binds your unique Oracle Net8 configuration into the file, which is used by the Oracle driver to access local and remote Oracle databases.

The Oracle 8 driver requires the shared library `libclntsh.so`, which is built by the Oracle script `genclntsh`. The `genclntsh` script provided by Oracle causes an error resulting from the undefined symbol `slpmprodstab`. Oracle 8 users must therefore use the `genclntsh8` script provided with Connect ODBC to build a replacement `libclntsh.so`. This script, in the `scr/oracle` directory, places the new `libclntsh.so` in `../lib`, which is your `$ODBC_HOME/lib` directory; it does not overwrite the original `libclntsh.so` in the `$ORACLE_HOME/lib` directory.

Before you build the Oracle Net8 shared library, install Oracle and set the environment variable `ORACLE_HOME` to the directory where you installed Oracle.

The following command builds the Oracle Net8 shared library:

```
genclntsh8
```

Macintosh

The Oracle SQL*Net 2.3.2.0.3 product is required to access remote Oracle databases.

Other system requirements are:

- 8 MB of memory (16 MB recommended)
- MacTCP version 1.1 or greater, or, for System 7.5, OpenTransport 1.1 or later if using TCP-IP as the transport protocol

Configuring Data Source

In the UNIX environment, there is no ODBC Administrator. To configure a data source in the UNIX environment, you must edit the system information file using the attributes in [Table 5-5](#). You must also edit this file to perform a translation. For information about this file, see [“The UNIX Environment”](#) on page 149.

To configure an Oracle data source:

1. Start the ODBC Administrator to display a list of data sources.
2. If you are configuring an existing data source, select the data source name and click **Configure** to display the **ODBC Oracle Driver Setup** dialog box (if you are using Apple's ODBC Driver Manager on the Macintosh, this button is labeled **Modify**).

If you are configuring a new data source, click **Add** to display a list of installed drivers. Select the Oracle driver of your choice and click **Finish** to display the **ODBC Oracle Driver Setup** dialog box.

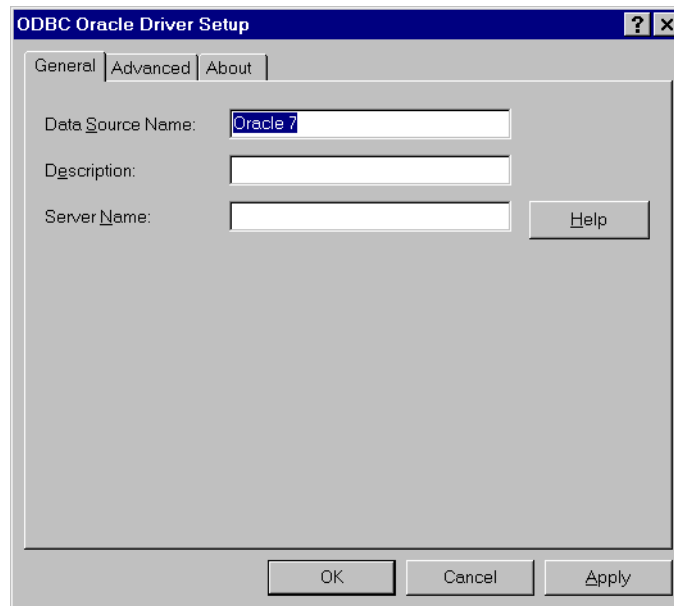


Figure 5-6: The **ODBC Oracle Driver Setup** dialog box.

3. Specify values as follows; then, click **Apply**:

Apply is not available on the Macintosh. Clicking **OK** saves the values.

Data Source Name: A string that identifies this Oracle data source configuration in the system information. Examples include “Accounting” or “Oracle-Serv1.”

Description: An optional long description of a data source name. For example, “My Accounting Database” or “Oracle on Server number 1.”

Server Name: The client connection string designating the server and database to be accessed. The information required varies depending on the client driver you are using. The format of the connection string is described in “Connecting to a Data Source Using a Connection String” on page 86.

4. Click the **Advanced** tab to configure additional, optional settings for the data source.

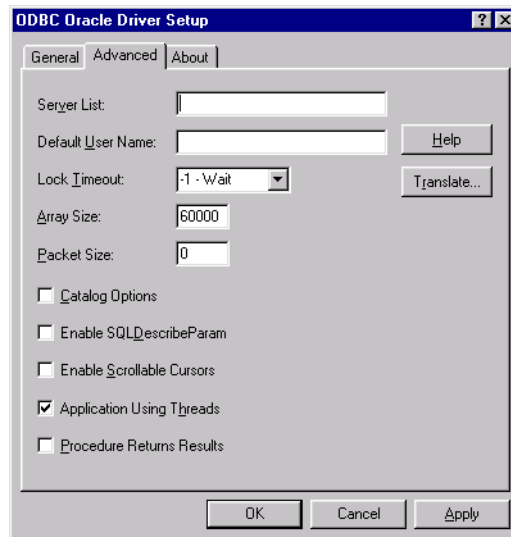


Figure 5-7: The **Advanced** tab of the **ODBC Oracle Driver Setup** dialog box.

5. Specify values as follows; then, click **Apply**:

Server List: The list of client connection strings that will appear in the logon dialog box. Separate the strings with commas. If the client connection string contains a comma, enclose it in quotation marks; for example, “Serv,1”, “Serv,2”, “Serv,3.”

Default User Name: The default user name used to connect to your Oracle database. A default user name is required only if security is enabled on your database. Your ODBC application may override this value or you may override this value in the logon dialog box or connection string.

Lock Timeout (Oracle 7 Only): A value of 0 or -1 that specifies whether Oracle should wait for a lock to be freed before raising an error when

processing a Select...For Update statement. Values can be -1 (wait forever) or 0 (do not wait). The default is -1.

Array Size: The number of bytes the driver uses for fetching multiple rows. Values can be an integer from 0 to 65536; the default is 60000. Larger values increase throughput by reducing the number of times the driver fetches data across the network. Smaller values increase response time, as there is less of a delay waiting for the server to transmit data.

Isolation Level (Oracle 8 only): The default isolation level for concurrent transactions. `SQL_TXN_READ_COMMITTED` and `SQL_TXN_READ_SERIALIZABLE` are the values. The default is `SQL_TXN_READ_COMMITTED`.

Packet Size (Oracle 7 Only): A value that controls the packet size for TCP/IP connections. Specify one of the following packet sizes: 1024, 2048, 4096, or 8192. Any other value is ignored.

The Packet Size option is used only when the connection string specified in the Server Name option is T for TCP/IP as the driver prefix. See the ServerName option described in [Table 5-5](#).

Catalog Options: Check this box if you want the result column `REMARKS` for the catalog functions `SQLTables` and `SQLColumns`, and `COLUMN_DEF` for the catalog function `SQLColumns` to have meaning for Oracle. The default is unchecked, which returns `SQL_NULL_DATA` for the result column `COLUMN_DEF` and `REMARKS` columns. Checking this box reduces the performance of your queries.

Enable SQLDescribeParam: Check this box to enable the `SQLDescribeParam` function, which results in all parameters being described with a data type of `SQL_VARCHAR`. This option should be checked when using Microsoft Remote Data Objects (RDO) to access data.

Enable Scrollable Cursors: Check this box to enable scrollable cursors for the data source. Both Keyset and Static cursors are enabled. This option may need to be checked when using Microsoft Foundation Classes for database access.

Application Using Threads: A setting that ensures that the driver works with multi-threaded applications. You can clear this check box when using the driver with single-threaded applications. Turning off this setting avoids additional processing required for ODBC thread-safety standards.

When Application Using Threads is enabled, SQLGetInfo(SQL_ASYNC_MODE) returns SQL_AM_NONE, SQLSetConnectAttr(SQL_ATTR_ASYNC_ENABLE) returns “optional feature not implemented,” and SQLSet/GetStmtAttr(SQL_ATTR_ASYNC_ENABLE) returns “optional feature not implemented.” Asynchronous execution is not supported by the Oracle client in a multi-threaded environment.

IDL does not support multi-threading.

Procedure Returns Results (Windows only): Check this box to enable the driver to return result sets from stored procedures/functions. If this option is on and you execute a stored procedure that does not return result sets, you will incur a small performance penalty. See “[Stored Procedure Results](#)” on page 107.

Translate: Click **Translate** to display the **Select Translator** dialog box, which lists the translators specified in the ODBC Translators section of the system information. INTERSOLV provides a translator named INTERSOLV OEM ANSI that translates your data from the IBM PC character set to the ANSI character set.

Select a translator; then, click **OK** to close this dialog box and perform the translation.

6. Click **OK** or **Cancel**. If you click **OK**, the values you have specified become the defaults when you connect to the data source. You can change these defaults by using this procedure to reconfigure your data source. You can override these defaults by connecting to the data source using a connection string with alternate values.

Connecting to a Data Source Using a Logon Dialog Box

Some ODBC applications display a logon dialog box when you are connecting to a data source. In these cases, the data source name has already been specified. For Oracle, the dialog box is as follows:

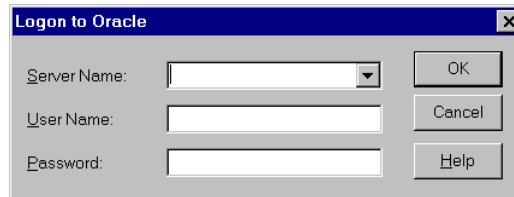


Figure 5-8: The **Logon to Oracle** dialog box.

In this dialog box, do the following:

1. Type the client connection string of the computer containing the Oracle database tables you want to access or select the string from the **Server Name** drop-down list box, which displays the names you specified in the setup dialog box.
2. If required, type your Oracle user name.
3. If required, type your Oracle password.
4. Click **OK** to log on to the Oracle database installed on the server you specified and to update the values in the system information.

Note

(Windows only) Oracle has a feature that allows you to connect to Oracle via the operating system user name and password. To connect, use a slash (/) for the user name and leave the password blank. To configure the Oracle server/client, refer to the Oracle server documentation.

Connecting to a Data Source Using a Connection String

If your application requires a connection string to connect to a data source, you must specify the data source name that tells the driver which section in the system information to use for the default connection information. Optionally, you may specify *attribute=value* pairs in the connection string to override the default values

stored in the system information. These values are not written to the system information.

You can specify either long or short names in the connection string. The connection string has the form:

```
DSN=data_source_name[;attribute=value  
[;attribute=value]...]
```

An example of a connection string for Oracle is:

```
DSN=Accounting;SRVR=X:QESRVR;UID=JOHN;PWD=XYZZY
```

If the server name contains a semicolon, enclose it in quotation marks:

```
DSN=Accounting;SRVR="X:QE;SRVR";UID=JOHN;PWD=XYZZY
```

[Table 5-5](#) gives the long and short names for each attribute, as well as a description.

To configure a data source in the UNIX environment, you must edit the system information file. This file accepts only long names for attributes. For information about this file, see [“The UNIX Environment”](#) on page 149.

The defaults listed in the table are initial defaults that apply when no value is specified in either the connection string or in the data source definition in the system

information. If you specified a value for the attribute when configuring the data source, that value is your default.

Attribute	Description
ApplicationUsingThreads (AUT)	<p>ApplicationUsingThreads={0 1}. Ensures that the driver works with multi-threaded applications. The default is 1, which makes the driver thread-safe. When using the driver with single-threaded applications, you may set this option to 0 to avoid additional processing required for ODBC thread-safety standards.</p> <p>When you specify ApplicationUsingThreads=1, SQLGetInfo(SQL_ASYNC_MODE) returns SQL_AM_NONE, SQLSetConnectAttr(SQL_ATTR_ASYNC_ENABLE) returns “optional feature not implemented,” and SQLSet/GetStmtAttr(SQL_ATTR_ASYNC_ENABLE) returns “optional feature not implemented.” Asynchronous execution is not supported by the Oracle client in a multi-threaded environment.</p> <p>Note - IDL does not support multi-threading.</p>
ArraySize (AS)	<p>The number of bytes the driver uses for fetching multiple rows. Values can be an integer from 0 to 65,536. The initial default is 60,000. Larger values increase throughput by reducing the number of times the driver fetches data across the network. Smaller values increase response time, as there is less of a delay waiting for the server to transmit data.</p>
CatalogOptions (CO)	<p>CatalogOptions={0 1}. Specifies whether the result column REMARKS for the catalog functions SQLTables and SQLColumns and COLUMN_DEF for the catalog function SQLColumns have meaning for Oracle. If you want to obtain the actual default value, set CO=1. The default is 0.</p>
DataSourceName (DSN)	<p>A string that identifies an Oracle data source configuration in the system information. Examples include “Accounting” or “Oracle-Serv1.”</p>

Table 5-5: : Oracle Connection String Attributes

Attribute	Description
DefaultIsolation Level (DIL) Oracle 8 Only	DefaultIsolationLevel=string values. Specifies the default isolation level for concurrent transactions. The values are SQL_TXN_READ_COMMITTED and SQL_TXN_READ_SERIALIZABLE. The default is SQL_TXN_READ_COMMITTED.
EnableDescribe Param (EDP)	EnableDescribeParam={0 1}. Enables the ODBC API function SQLDescribeParam, which results in all parameters being described with a data type of SQL_VARCHAR. This option should be set to 1 when using Microsoft Remote Data Objects (RDO) to access data. The default is 0.
EnableScrollable Cursors (ESC)	EnableScrollableCursors={0 1}. Enables scrollable cursors for the data source. Both Keyset and Static cursors are enabled. This option may need to be set to 1 when using Microsoft Foundation Classes for database access. The default is 0.
LockTimeOut (LTO) Oracle 7 Only	A value that specifies whether Oracle should wait for a lock to be freed before raising an error when processing a Select...For Update statement. Values can be -1 (wait forever, the initial default) or 0 (do not wait).
LogonID (UID)	The logon ID (user name) that the application uses to connect to your Oracle database. A logon ID is required only if security is enabled on your database. If so, contact your system administrator to get your logon ID. To use your operating system user name, see “Connecting to a Data Source Using a Logon Dialog Box” on page 85.
PacketSize (PS) Oracle 7 Only	PacketSize={1024 2048 4096 8192}. A value that controls the packet size for TCP/IP connections. Any values other than 1024, 2048, 4096, or 8192 are ignored. This value is used only when the ServerName attribute (described above) is set to T for TCP/IP.
Password (PWD)	The password that the application uses to connect to your Oracle database. To use your operating system password, see “Connecting to a Data Source Using a Logon Dialog Box” on page 85.

Table 5-5: : Oracle Connection String Attributes

Attribute	Description
ProcedureRet Results (PRR) (Windows only)	ProcedureRetResults={0 1}. Values are Off (0) and On (1). The default is 0. When the option is on, the driver will return result sets from stored procedures/functions. If this option is on and you execute a stored procedure that does not return result sets, you will incur a small performance penalty. See “Stored Procedure Results” on page 107.

Table 5-5: : Oracle Connection String Attributes

Attribute	Description
ServerName (SRVR)	<p>The client connection string designating the server and database to be accessed. The information required varies depending on the client driver that you are using.</p> <p>For Oracle 7 remote servers, the SQL*Net connection string has the following form:</p> <p><i>driver_prefix:computer_name[:sid]</i></p> <p><i>driver_prefix</i> identifies the network protocol being used. The driver prefix can be as follows: P (named pipes), X (SPX), B (NetBIOS), T (TCP/IP), D (DECNet), A (Oracle Async), AT (AppleTalk), or TNS (SQL*Net 2.0). Check your Oracle documentation for other protocols.</p> <p><i>computer_name</i> is the name of the Oracle Listener on your network.</p> <p><i>sid</i> is the Oracle System Identifier and refers to the instance of Oracle running on the host. This item is required when connecting to systems that support more than one instance of an Oracle database.</p> <p>For local servers, the SQL*Net connection string has the form: <i>database_name</i></p> <p><i>database_name</i> identifies your Oracle database.</p> <p>If the SQL*Net connection string contains semicolons, enclose it in quotation marks. See your SQL*Net documentation for more information.</p> <p>Oracle 8</p> <p>For Oracle 8 remote servers, the Net8 Client connection string has the following form:</p> <p>TNSNAME</p> <p><i>TNSNAME</i> is the alias name of the Oracle Listener on your network.</p> <p>If the Net8 Client connection string contains semicolons, enclose it in quotation marks. See your Net8 Client documentation for more information.</p>

Table 5-5: : Oracle Connection String Attributes

Oracle Data Types

The following table shows how the Oracle data types are mapped to the standard ODBC data types.

Oracle	ODBC
Char	SQL_CHAR
Date	SQL_TYPE_TIMESTAMP
Long	SQL_LONGVARCHAR
Long Raw	SQL_LONGVARBINARY
Number	SQL_DOUBLE
Number(p,s)	SQL_DECIMAL
Raw	SQL_VARBINARY
Varchar2	SQL_VARCHAR

Table 5-6: : Oracle Data Types

Oracle 8

The following table shows how the Oracle 8 data types are mapped to the standard ODBC data types. These are in addition to the Oracle data types described above.

Oracle 8	ODBC
Bfile	SQL_LONGVARBINARY*
Blob	SQL_LONGVARBINARY
Clob	SQL_LONGVARCHAR
* Read-Only	

Table 5-7: : Oracle 8 Data Types

The Oracle 8 driver does not support any Abstract Data Types. When the driver encounters an Abstract Data Type during data retrieval, it will return an Unknown Data Type error (SQL State HY000). It also does not support asynchronous operations, due to constraints in the current Oracle 8 client.

Stored Procedure Results

When the option Procedure Returns Results is active, the driver returns result sets from stored procedures/functions. In addition, `SQLGetInfo(SQL_MULT_RESULTS_SETS)` will return “Y” and `SQLGetInfo(SQL_BATCH_SUPPORT)` will return `SQL_BS_SELECT_PROC`. If this option is on and you execute a stored procedure that does not return result sets, you will incur a small performance penalty.

This feature requires that stored procedures be in a certain format. First, a package must be created to define all of the cursors used in the procedure, then the procedure can be created using the new cursor. For example:

```
Create or replace package GEN_PACKAGE as CURSOR G1 is select
CHARCOL from GTABLE2;
type GTABLE2CHARCOL is ref cursor return G1%rowtype;
end GEN_PACKAGE;

Create or replace procedure GEN_PROCEDURE1 (rset IN OUT
GEN_PACKAGE.GTABLE2CHARCOL, icol INTEGER) as begin
open rset for select CHARCOL from GTABLE2 where INTEGERCOL <= icol
order by INTEGERCOL;
end;
```

For more information consult your Oracle SQL manual.

Isolation and Lock Levels Supported

Oracle supports isolation level 1 (read committed) and isolation level 3 (serializable isolation—if the server version is Oracle 7.3 or greater or Oracle 8.x). Oracle supports record-level locking.

See [“Locking and Isolation Levels”](#) on page 152 for a discussion of these topics.

ODBC Conformance Level

The Oracle drivers support the functions listed in [Chapter 6, “ODBC API and Scalar Functions”](#). The drivers also support `SQLDescribeParam` if `EnableDescribeParam=1`. If `EnableScrollableCursors=1`, they support `SQLSetPos` as well as scrollable cursors with `SQLFetchScroll` and `SQLExtendedFetch`.

The Oracle drivers support the following X/Open level functions:

- `SQLProcedures`
- `SQLProcedureColumns`

- SQLPrimaryKeys
- SQLForeignKeys
- SQLTablePrivileges
- SQLColumnPrivileges
- SQLSetPos (SQL_ADD)

The drivers support the core SQL grammar.

Number of Connections and Statements Supported

The Oracle drivers support multiple connections and multiple statements per connection.

Connect ODBC for Sybase

Connect ODBC for Sybase (the “Sybase driver”) supports the SQL Server System 10, System 11, and Adaptive Server 11.5 and 11.9 database systems from Sybase in the Windows 9x, Windows NT, Macintosh, and UNIX environments. The driver supports the SQL Server 4.9.2 database system in the Windows 9x and Windows NT environments.

See the README file shipped with your INTERSOLV DataDirect product for the file name of the Sybase driver.

System Requirements

The following section lists requirements for all supported platforms.

Windows 9x and Windows NT

You must install the Sybase Open Client-Library (version 10.0.4 or higher for Intel systems, version 11.1.1 for Alpha systems) and the appropriate Sybase Net-Library to gain access to the Sybase server.

SQLEdit is a tool that allows you to define servers and adds them to SQL.INI.

SYBPING is a tool that is provided to test connectivity from your client workstation to the database server (servers that are added through SQLEdit). Use this tool to test your connection.

Set the environment variable SYBASE to the directory where you installed the Sybase Open Client. For example, set SYBASE=C:\SQL10. For Windows, set this environment variable in the **Control Panel** under **System**.

UNIX

Before you can use the System data source, you must have the Sybase Open Client Net-Libraries you plan to use installed on your workstation in the \$SYBASE source tree.

Set the environment variable SYBASE to the directory where you installed the System client. For example, for C-shell users, the following syntax is valid:

```
setenv SYBASE /databases/sybase
```

For Bourne- or Korn-shell users, the following syntax is valid:

```
SYBASE=/databases/sybase;export SYBASE
```

You must include the directory containing the System client-shared libraries in the environment variable `LD_LIBRARY_PATH` (on Solaris), `LIBPATH` (on AIX), and `SHLIB_PATH` (on HP-UX). For example, for C-shell users, the following syntax is valid:

```
setenv LD_LIBRARY_PATH /databases/sybase
/lib:$LD_LIBRARY_PATH
```

For Bourne- or Korn-shell users, the following syntax is valid:

```
LD_LIBRARY_PATH=/databases/sybase
/lib:$LD_LIBRARY_PATH;export LD_LIBRARY_PATH
```

In non-DCE environments, users should use the `ivsybxx` Sybase driver that requires the library `libct`. For DCE environments, users should use the `ivsybl1xx` Sybase driver that requires the Sybase 11.1 client library `libct_r`.

Macintosh

You must install the Sybase Open Client-Library, version 10.0.3 or higher, and the appropriate Sybase Net-Library to gain access to the Sybase server. Other system requirements are:

- 8 MB of memory
- MacTCP version 1.1 or greater or OpenTransport version 1.1 if using TCP

Double-click the **Sybase Config Control Panel** and select your interface file. See your System documentation for more information.

You can use `Sybping` to test the connection to the database server.

Configuring Data Sources

Note

In the UNIX environment, there is no ODBC Administrator. To configure a data source in the UNIX environment, you must edit the system information file using the attributes in [Table 5-8](#). You must also edit this file to perform a translation. For information about this file, see [“The UNIX Environment”](#) on page 149.

To configure a Sybase data source:

1. Start the ODBC Administrator to display a list of data sources.
2. If you are configuring an existing data source, select the data source name and click **Configure** to display the **ODBC Sybase Driver Setup** dialog box.

If you are configuring a new data source, click **Add** to display a list of installed drivers. Select the Sybase driver and click **Finish** to display the **ODBC Sybase Driver Setup** dialog box.



Figure 5-9: ODBC Sybase Driver Setup dialog box.

3. Specify values as follows; then, click **Apply**:

Note

Apply is not available on the Macintosh. Clicking **OK** saves the values.

Data Source Name: A string that identifies this Sybase data source configuration in the system information. Examples include “Accounting” or “Sys10-Serv1.”

Description: An optional long description of a data source name. For example, “My Accounting Database” or “System 10 on Server number 1.”

Server Name: The name of the server that contains the Sybase tables you want to access. If not supplied, the server name in the DSQUERY environment variable is used. On UNIX, the name of a server from your \$SYBASE/interfaces file.

Database Name: The name of the database to which you want to connect by default. If you do not specify a value, the default is the database defined by the system administrator for each user.

4. Click the **Advanced** tab to configure additional, optional settings for the data source.

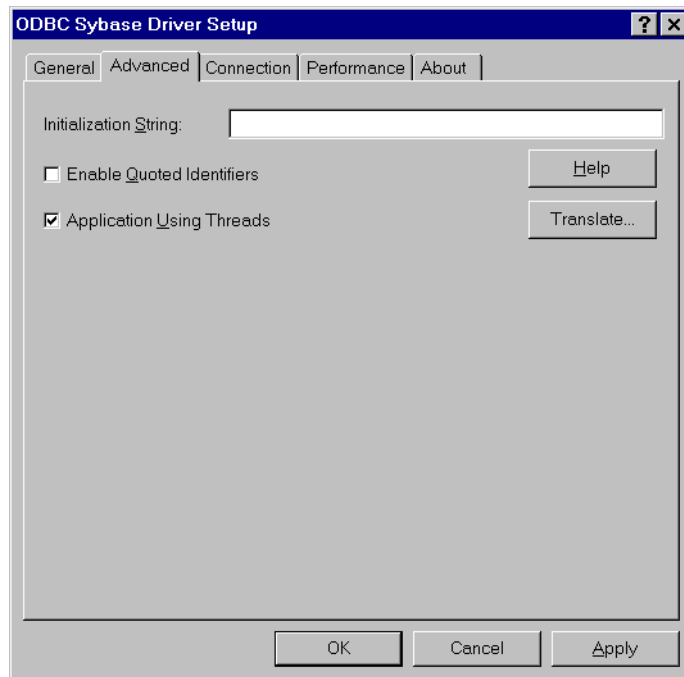


Figure 5-10: The **Advanced** tab of the **ODBC Sybase Driver Setup**.

5. Specify values as follows; then, click **Apply**:

Initialization String: Supports the running of Sybase commands at connect time. Multiple commands must be separated by semicolons.

Default Buffer Size for Long Columns (in Kb): An integer value that specifies, in 1024-byte multiples, the maximum length of data fetched from a

TEXT or IMAGE column. The default is 1,024 kilobytes. You will need to increase this value if the total size of any long data exceeds 1 megabyte.

Enable Quoted Identifiers: Allows support of quoted identifiers in System 10 or System 11 servers.

Application Using Threads: Ensures that the driver works with multi-threaded applications. You can clear this check box when using the driver with single-threaded applications. Turning off this setting avoids additional processing required for ODBC thread safety standards.

Note

IDL does not support multi-threading.

Cursor Positioning for raiserror: A value of 0 or 1 that specifies when the error is returned and where the cursor is positioned when raiserror is encountered.

When set to 0 (the default), raiserror is handled separately from surrounding statements. The error is returned when raiserror is processed via SQLExecute, SQLExecDirect, or SQLMoreResults. The result set is empty.

When set to 1 (MS compatible), raiserror is handled with the next statement. The error is returned when the next statement is processed; the cursor is positioned on the first row of subsequent result set. This could result in multiple raiserrors being returned on a single execute.

Translate: Click **Translate** to display the **Select Translator** dialog box, which lists the translators specified in the ODBC Translators section of the system information. INTERSOLV provides a translator named INTERSOLV OEM ANSI that translates your data from the IBM PC character set to the ANSI character set.

Select a translator; then, click **OK** to close this dialog box and perform the translation.

- Click the **Connection** tab to configure optional data source settings.

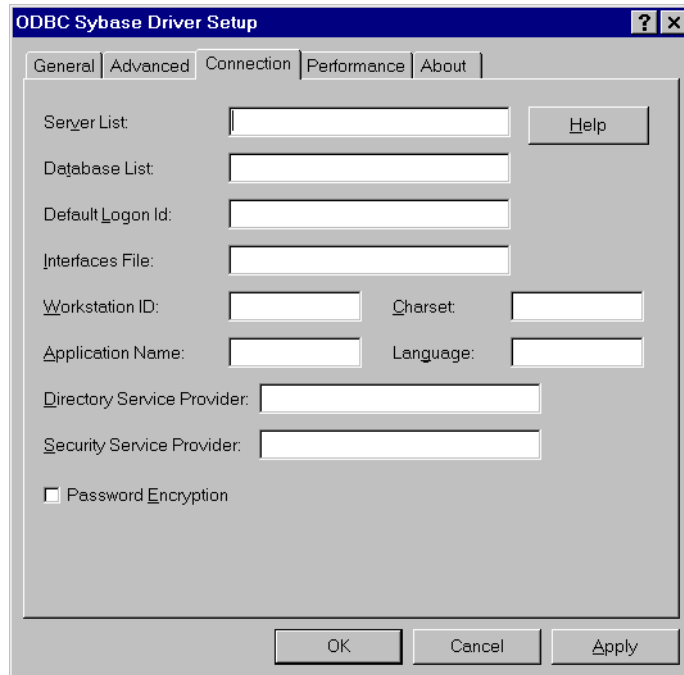


Figure 5-11: The **Connection** tab of the **ODBC Sybase Driver Setup**.

- Specify values as follows; then, click **Apply**:

Server List: The list of servers that appear in the logon dialog box. Separate the server names with commas.

Database List: The databases that appear in the logon dialog box. Separate the names with commas.

Default Logon ID: The default logon ID used to connect to your Sybase database. This ID is case-sensitive. A logon ID is required only if security is enabled for the database you are connecting to. Your ODBC application may override this value or you can override this value in the logon dialog box or connection string.

Interfaces File: The path name of the interfaces file. The default is the normal Sybase interfaces file.

Workstation ID: The workstation ID used by the client.

Charset: The name of a character set corresponding to a subdirectory in `$SYBASE/charsets`. The default is the setting on the Sybase server.

Application Name: The name used by Sybase to identify your application.

Language: The national language corresponding to a subdirectory in `$SYBASE/locales`. The default is English.

Directory Service Provider: A string that indicates which Directory Service Provider the Sybase Open Client uses when connecting with this data source. The available Directory Service Providers can be found using the OpenClient/OpenServer Configuration Utility that is installed with Sybase Open Client version 11.1 or higher. If the client is not using Open Client version 11.1 or higher, this option is ignored.

Note

Directory Service Provider is not available on the Macintosh.

Security Service Provider: A string that indicates which Security Service Provider the Sybase Open Client uses when connecting with this data source. The available Security Service Providers can be found using the OpenClient/OpenServer Configuration Utility that is installed with Sybase Open Client version 11.1 or higher. If the client is not using Open Client version 11.1 or higher, this option is ignored.

Note

Security Service Provider is not available on the Macintosh.

Password Encryption: A value that determines whether password encryption can be performed from the Open Client Library to the server. Checking this box enables password encryption.

- Click the **Performance** tab to configure performance settings for this data source.

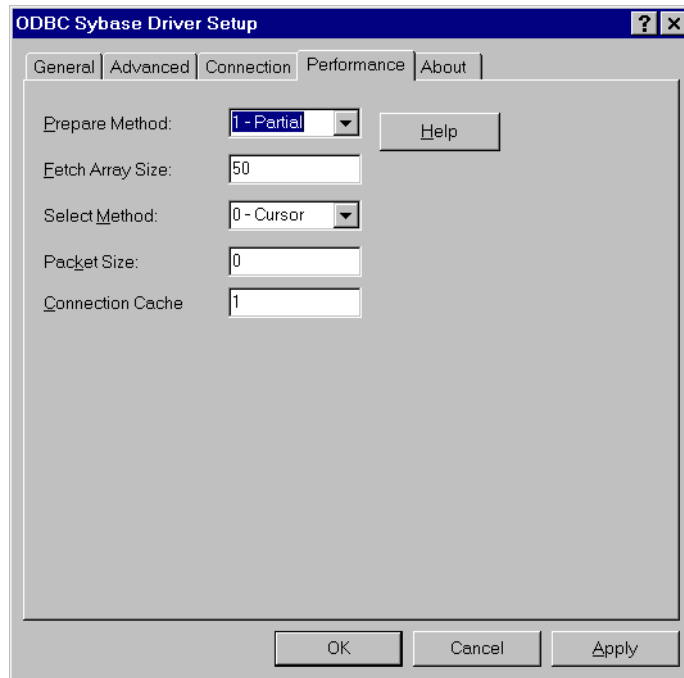


Figure 5-12: The **Performance** tab of the **ODBC Sybase Driver Setup**.

- Specify values as follows; then, click **Apply**:

Prepare Method: A value of 0, 1, or 2 that determines whether stored procedures are created on the server for every call to SQLPrepare.

When set to 0, stored procedures are created for every call to SQLPrepare. This setting can result in bad performance when processing static statements.

When set to 1, the initial default, the driver creates stored procedures only if the statement contains parameters. Otherwise, the statement is cached and run directly at SQLExecute time.

When set to 2, the driver never creates stored procedures. This setting is ignored when connected to Sybase 4.9.2 servers.

Fetch Array Size: The number of rows the driver retrieves when fetching from the server. This is not the number of rows given to the user. The default is 50 rows.

Select Method: A value of 0 or 1 that determines whether database cursors are used for Select statements. When set to 0, the default, database cursors are used; when set to 1, Select statements are run directly without using database cursors. A setting of 1 limits the data source to one active statement. This setting is ignored when connected to Sybase 4.9.2 servers.

Packet Size: A value of -1, 0, or x that determines the number of bytes per network packet transferred from the database server to the client. The correct setting of this attribute can improve performance.

When set to 0, the default, the driver uses the default packet size as specified in the Sybase server configuration.

When set to -1, the driver computes the maximum allowable packet size on the first connect to the data source and saves the value in the system information.

When set to x , an integer from 1 to 10, which indicates a multiple of 512 bytes (for example, 6 means to set the packet size to $6 * 512 = 3072$ bytes).

To take advantage of this connection attribute, you must configure the Sybase server for a maximum network packet size greater than or equal to the value you specified for PacketSize. For example,

```
sp_configure "maximum network packet size", 5120
reconfigure
Restart Sybase Server
```

Note that the ODBC specification identifies a connect option, `SQL_PACKET_SIZE`, that offers this same functionality. To avoid conflicts with applications that may set both the connection string attribute and the ODBC connect option, they have been defined as mutually exclusive. If PacketSize is specified, you will receive a message "Driver Not Capable" if you attempt to call `SQL_PACKET_SIZE`. If you do not set PacketSize, then application calls to `SQL_PACKET_SIZE` are accepted by the driver.

Connection Cache: A value that determines the number of connections that the connection cache can hold. The default Connection Cache setting is 1. To set the connection cache, you must set the Select Method option to 1 - Direct. Increasing the connection cache may increase performance of some applications but requires additional database resources.

10. Click **OK** or **Cancel**. If you click **OK**, the values you have specified become the defaults when you connect to the data source. You can change these defaults by using this procedure to reconfigure your data source. You can override these defaults by connecting to the data source using a connection string with alternate values.

Connecting to a Data Source Using a Logon Dialog Box

Some ODBC applications display a Logon dialog box when you are connecting to a data source. In these cases, the data source name has already been specified. For Sybase, the dialog box is as follows:

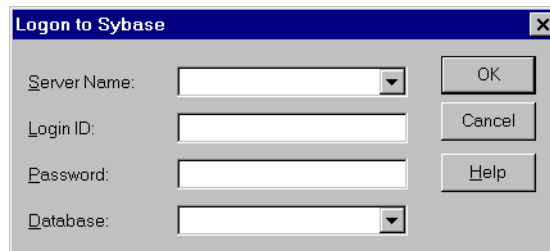


Figure 5-13: The **Logon to Sybase** dialog box.

In this dialog box, do the following:

1. Type the case-sensitive name of the server containing the Sybase database tables you want to access or select the name from the **Server Name** drop-down list, which displays the server names you specified in the **ODBC Sybase Driver Setup** dialog box.
2. If required, type your case-sensitive login ID.
3. If required, type your case-sensitive password for the system.
4. Type the name of the database you want to access (case-sensitive) or select the name from the **Database** drop-down list, which displays the names you specified in the **ODBC Sybase Driver Setup** dialog box.
5. Click **OK** to complete the logon and to update the values in the system information.

Connecting to a Data Source Using a Connection String

If your application requires a connection string to connect to a data source, you must specify the data source name that tells the driver which section in the system information to use for the default connection information. Optionally, you may specify *attribute=value* pairs in the connection string to override the default values stored in the system information. These values are not written to the system information.

You can specify either long or short names in the connection string. The connection string has the form:

```
DSN=data_source_name[;attribute=value
[;attribute=value]...]
```

An example of a connection string for Sybase is:

```
DSN=SYS10 TABLES;SRVR=QESRVR;DB=PAYROLL;UID=JOHN;PWD=XYZZY
```

The following table gives the long and short names for each attribute, as well as a description.

To configure a data source in the UNIX environment, you must edit the system information file. This file accepts only long names for attributes. For information about this file, see [“The UNIX Environment”](#) on page 149.

The defaults listed in the table are initial defaults that apply when no value is specified in either the connection string or in the data source definition in the system information. If you specified a value for the attribute when configuring the data source, that value is your default.

Attribute	Description
ApplicationName (APP)	The name used by Sybase to identify your application.
ApplicationUsingThreads (AUT)	ApplicationUsingThreads={0 1}. Ensures that the driver works with multi-threaded applications. The default is 1, which makes the driver thread-safe. When using the driver with single-threaded applications, you may set this option to 0 to avoid additional processing required for ODBC thread safety standards. Note - IDL does not support multi-threading.

Table 5-8: : Sybase Connection String Attributes

Attribute	Description
ArraySize (AS)	The number of rows the driver retrieves from the server for a fetch. This is not the number of rows given to the user. This increases performance by reducing network traffic. The initial default is 50 rows.
Charset (CS)	The name of a character set corresponding to a subdirectory in \$SYBASE/charsets.
CursorCacheSize (CCS)	The number of connections that the connection cache can hold. The initial default value for CursorCacheSize is 1 (one cursor). To set the connection cache, you must set the SelectMethod attribute to 1. Increasing the connection cache may increase performance of some applications but requires additional database resources.
Database (DB)	The name of the database to which you want to connect.
DataSourceName (DSN)	A string that identifies a single connection to a Sybase database. Examples include "Accounting" or "Sys10-Serv1."
DefaultLongData BuffLen (DLDBL)	An integer value that specifies, in 1024-byte multiples, the maximum length of data fetched from a TEXT or IMAGE column. The default is DefaultLongDataBuffLen=1024. You will need to increase this value if the total size of any long data exceeds 1 megabyte.
DirectoryService Provider (DSP)	A string that indicates which Directory Service Provider the Sybase Open Client uses when connecting with this data source. The available Directory Service Providers can be found using the OpenClient/OpenServer Configuration Utility that is installed with Sybase Open Client version 11.1 or higher. If the client is not using Open Client version 11.1 or higher, this option is ignored. Directory Service Provider is <i>not</i> available on the Macintosh.
EnableQuoted Identifiers (EQI)	EnableQuotedIdentifiers={0 1}. Specify 1 to allow support of quoted identifiers. The default is 0.

Table 5-8: : Sybase Connection String Attributes

Attribute	Description
InitializationString (IS)	InitializationString={<Sybase set commands>;...}. Supports the execution of Sybase commands at connect time. Multiple commands must be separated by semicolons.
InterfacesFile (IFILE)	The path name to the interfaces file.
Language (LANG)	The national language corresponding to a subdirectory in \$SYBASE/locales.
LogonID (UID)	The default logon ID used to connect to your Sybase database. This ID is case-sensitive. A logon ID is required only if security is enabled on your database. If so, contact your system administrator to get your logon ID.
OptimizePrepare (OP)	<p>OptimizePrepare={0 1 2}. This attribute determines whether stored procedures are created on the server for every call to SQLPrepare.</p> <p>When set to 0, stored procedures are created for every call to SQLPrepare. This setting can result in bad performance when processing static statements.</p> <p>When set to 1, the initial default, the driver creates stored procedures only if the statement contains parameters. Otherwise, the statement is cached and run directly at SQLExecute time.</p> <p>When set to 2, the driver never creates stored procedures. This attribute is ignored for Sybase 4.9.2 servers.</p>

Table 5-8: : Sybase Connection String Attributes

Attribute	Description
PacketSize (PS)	<p>PacketSize={-1 0 x}. This attribute determines the number of bytes per network packet transferred from the database server to the client. The correct setting of this attribute can improve performance.</p> <p>When set to 0, the initial default, the driver uses the default packet size as specified in the Sybase server configuration.</p> <p>When set to -1, the driver computes the maximum allowable packet size on the first connect to the data source and saves the value in the system information.</p> <p>When set to x, an integer from 1 to 10, which indicates a multiple of 512 bytes (for example, PacketSize=6 means to set the packet size to $6 * 512 = 3072$ bytes).</p> <p>To take advantage of this connection attribute, you must configure the Sybase server for a maximum network packet size greater than or equal to the value you specified for PacketSize. For example:</p> <pre>sp_configure "maximum network packet size", 5120 reconfigure Restart Sybase Server</pre> <p>Note - The ODBC specification specifies a connect option, SQL_PACKET_SIZE, that offers this same functionality. To avoid conflicts with applications that may set both the connection string attribute and the ODBC connect option, they are defined as mutually exclusive. If PacketSize is specified, you will receive a message "Driver Not Capable" if you attempt to call SQL_PACKET_SIZE. If you do not set PacketSize, then application calls to SQL_PACKET_SIZE are accepted by the driver.</p>
Password (PWD)	A case-sensitive password.

Table 5-8: : Sybase Connection String Attributes

Attribute	Description
Password Encryption (PE)	<p>PasswordEncryption={0 1}. A value that determines whether password encryption can be performed from the Open Client Library to the server. When set to 0, the initial default, this cannot be done. When set to 1, password encryption is enabled.</p>
RaiseErrorPosition Behavior (REPB)	<p>RaiseErrorPositionBehavior={0 1}. A value that specifies when the error is returned and where the cursor is positioned when raiserror is encountered.</p> <p>When set to 0 (the default), raiserror is handled separately from surrounding statements. The error is returned when raiserror is processed via SQLExecute, SQLExecDirect, or SQLMoreResults. The result set is empty.</p> <p>When set to 1 (MS compatible), raiserror is handled with the next statement. The error is returned when the next statement is processed; the cursor is positioned on the first row of subsequent result set. This could result in multiple raiserrors being returned on a single execute.</p>
SecurityService Provider (SSP)	<p>A string that indicates which Security Service Provider the Sybase Open Client uses when connecting with this data source. The available Security Service Providers can be found using the OpenClient/OpenServer Configuration Utility that is installed with Sybase Open Client version 11.1 or higher. If the client is not using Open Client version 11.1 or higher, this option is ignored.</p> <p>Note - Security Service Provider is <i>not</i> available on the Macintosh.</p>

Table 5-8: : Sybase Connection String Attributes

Attribute	Description
SelectMethod (SM)	<p>SelectMethod={0 1}. This attribute determines whether database cursors are used for Select statements. When set to 0, the initial default, database cursors are used. In some cases performance degradation can occur when performing large numbers of sequential Select statements because of the amount of overhead associated with creating database cursors.</p> <p>When set to 1, Select statements are run directly without using database cursors. When set to 1, the data source is limited to one active statement.</p> <p>This attribute is ignored for Sybase 4.9.2 servers.</p>
ServerName (SRVR)	<p>The name of the server containing the Sybase tables you want to access. If not supplied, the initial default is the server name in the DSQUERY environment variable. On UNIX, the name of a server from your \$SYBASE/interfaces file.</p>
WorkstationID (WKID)	<p>The workstation ID used by the client.</p>

Table 5-8: : Sybase Connection String Attributes

Data Types

The following table shows how the Sybase data types are mapped to the standard ODBC data types.

Sybase	ODBC
binary	SQL_BINARY
bit	SQL_BIT
char	SQL_CHAR
datetime	SQL_TYPE_TIMESTAMP
* Not supported with Sybase 4.9.2 servers.	

Table 5-9: Sybase Data Types

Sybase	ODBC
decimal*	SQL_DECIMAL
float	SQL_FLOAT
image	SQL_LONGVARBINARY
int	SQL_INTEGER
money	SQL_DECIMAL
numeric*	SQL_NUMERIC
real	SQL_REAL
smalldatetime	SQL_TYPE_TIMESTAMP
smallint	SQL_SMALLINT
smallmoney	SQL_DECIMAL
sysname	SQL_VARCHAR
text	SQL_LONGVARCHAR
timestamp	SQL_VARBINARY
tinyint	SQL_TINYINT
varbinary	SQL_VARBINARY
varchar	SQL_VARCHAR
* Not supported with Sybase 4.9.2 servers.	

Table 5-9: Sybase Data Types

Isolation and Lock Levels Supported

Sybase supports isolation levels 0 (if the server version is 11 or higher), 1 (read committed, the default), and 3 (serializable). It supports page-level locking. See [“Locking and Isolation Levels”](#) on page 152 for a discussion of these topics.

ODBC Conformance Level

The Sybase driver supports the functions listed in [Chapter 6, “ODBC API and Scalar Functions”](#). In addition, the following functions are supported:

- SQLColumnPrivileges

- SQLForeignKeys
- SQLPrimaryKeys
- SQLProcedureColumns
- SQLProcedures
- SQLTablePrivileges

The driver supports the minimum SQL grammar.

Number of Connections and Statements Supported

The Sybase database system supports multiple connections and multiple statements per connection. If SelectMethod=1, Sybase data sources are limited to one active statement in manual commit mode.

Connect ODBC for Text

Connect ODBC for Text (the “Text driver”) supports ASCII text files in the Windows 9x, Windows NT, Macintosh, and UNIX environments. These files can be printed directly or edited with text editors or word processors, because none of the data is stored in a binary format.

See the `README` file shipped with your INTERSOLV DataDirect product for the file name of the text driver.

The Text driver executes SQL statements directly on the text files. The driver supports Insert statements, and inserts the record at the end of the file. You can execute Update and Delete statements conditionally.

System Requirements

Macintosh users who are accessing the same text file must have file sharing enabled.

Formats for Text Files

Some common formats for text files are listed in the following table.

Format	Description
Comma-separated values	Commas separate column values, and each line is a separate record. Column values can vary in length. These files often have the <code>.CSV</code> extension.
Tab-separated values	Tabs separate column values, and each line is a separate record. Column values can vary in length.
Character-separated values	Any printable character except single or double quotation marks can separate column values, and each line is a separate record. Column values can vary in length.

Table 5-10: : Common Text File Formats

Format	Description
Fixed	No character separates column values. Instead, values start at the same position and have the same length in each line. The values appear in fixed columns if you display the file. Each line is a separate record.
Stream	No character separates column values nor records. The table is one long stream of bytes.

Table 5-10: : Common Text File Formats

Comma-, tab-, and character-separated files are called character-delimited files because values are separated by a special character.

Configuring Data Sources

Note

In the UNIX environment, there is no ODBC Administrator. To configure a data source in the UNIX environment, you must edit the system information file using the attributes in [Table 5-13](#). You must also edit this file to perform a translation. For information about this file, see “[The UNIX Environment](#)” on page 149.

To configure a Text data source:

1. Start the ODBC Administrator to display a list of data sources.
2. If you are configuring an existing data source, select the data source name and click **Configure** to display the **ODBC Text Driver Setup** dialog box.

If you are configuring a new data source, click **Add** to display a list of installed drivers. Select the **Text** driver and click **Finish** to display the **ODBC Text Driver Setup** dialog box.

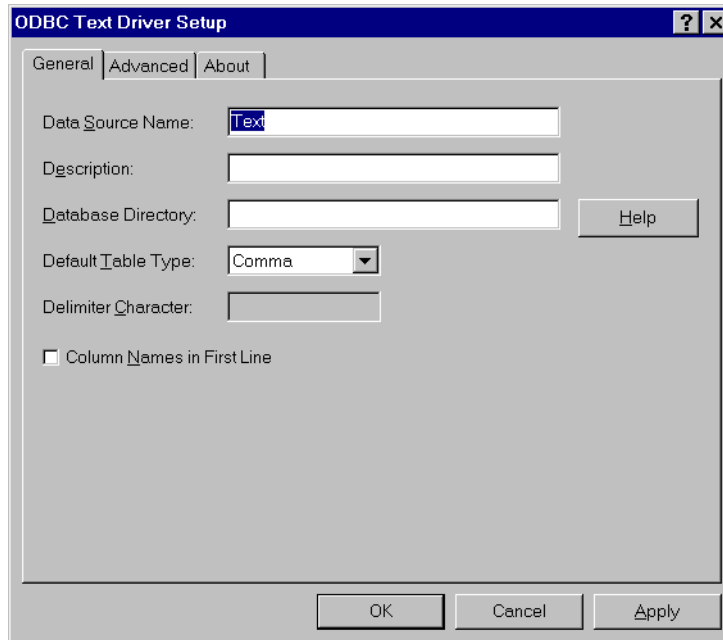


Figure 5-14: The **ODBC Text Driver Setup** dialog box.

3. Specify values as follows; then, click **Apply**:

Note

Apply is not available on the Macintosh. Clicking **OK** saves the values.

Data Source Name: A string that identifies this Text data source configuration in the system information. Examples include “Accounting” or “Text Files.”

Description: An optional long description of a data source name. For example, “My Accounting Files” or “My Text Files in the Accounting Directory.”

Database Directory: The directory in which the text files are stored. If none is specified, the current working directory is used.

Note

On the Macintosh, click **Select Directory**.

Default Table Type: The type of text file: comma-separated, tab-separated, character-separated, fixed length, or stream. This value tells the driver the default type, which is used when creating a new table and opening an undefined table.

Delimiter Character: The character used as a delimiter for character-separated files. It can be any printable character. The default is a comma (,).

Column Names in First Line: Select this check box to tell the driver to look for column names in the first line of the file.

Note

The **Default Table Type**, **Delimiter Character**, and **Column Names in First Line** settings apply only to tables not previously defined. These fields also determine the attributes of new tables created with the Create Table statement.

4. Click the **Advanced** tab to configure additional, optional settings for the data source.

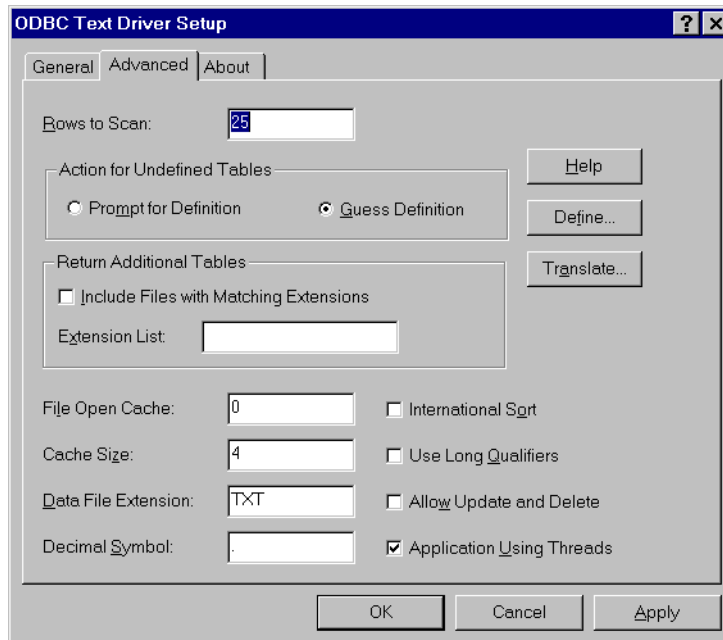


Figure 5-15: The **Advance** tab of the **ODBC Text Driver Setup** dialog box.

5. Specify values as follows; then, click **Apply**:

Rows to Scan: The number of rows in a text file that the driver scans to determine the data types in the file. If the value is 0, all rows in the file are scanned. The default is 25.

Note

The **Rows to Scan** setting applies only to tables *not* previously defined. This field also determines the attributes of new tables created with the Create Table statement.

Action for Undefined Tables: Two radio buttons that indicate what action the driver should take when it encounters a file that has not been defined. Select the **Prompt for Definition** radio button, if you want the driver to prompt the user when it encounters a file whose format is not defined. Otherwise, select

the **Guess Definition** radio button; in this case, the driver guesses the file's format.

Note

On the Macintosh, select **Guess** or **Prompt for Definition** from the pop-up menu.

Return Additional Tables: Select this check box to tell the driver to return files with a given extension in addition to the files specified in the **Data File Extension** field. In **Extension List**, specify a comma-separated list of the extensions. To have files with no extensions returned, specify NONE. For example, if some of your files have the extensions TXT and CSV and others have no extension, specify TXT, CSV, NONE.

By default, when an application requests a list of tables, only files that have been defined are returned.

Note

On the Macintosh, the **Return Additional Tables** functionality is found on the **Mac File Types** tab (see [“On the Macintosh”](#) on page 134).

File Open Cache: An integer value that specifies the maximum number of unused file opens to cache. For example, the value 4 specifies that when a user opens and closes four tables, the tables are not actually closed. The driver keeps them open so that if another query uses one of these tables, the driver does not have to perform another open, which is expensive. The advantage of file open caching is increased performance. The disadvantage is that a user who specifies file locking on open may get a locking conflict even though no one appears to have the file open. The default is 0, which means no file open caching.

Cache Size: The number of 64 KB blocks the driver uses to cache database records. The higher the number of blocks, the better the performance. The maximum number of blocks you can set depends on the system memory available. If the cache size is greater than 0, when browsing backwards, you will not be able to see updates made by other users until you run the Select statement again. The default is 4.

Data File Extension: Specifies the file extension to use for data files. The default **Data File Extension** setting is TXT. The **Data File Extension** setting cannot be greater than three characters. The **Data File Extension** setting is

used for all Create Table statements. Sending a Create Table using an extension other than the **Data File Extension** setting causes an error.

In other SQL statements, such as Select or Insert, users can specify an extension other than the **Data File Extension** setting. The **Data File Extension** setting is used when no extension is specified.

Decimal Symbol: A setting that specifies the decimal separator used when data is stored (may be a comma or period). The international decimal symbol (.) must be used in DML statements and parameter buffers.

International Sort: A setting to indicate the order in which records are retrieved when you issue a Select statement with an Order By clause. Clear this box to use ASCII sort order (the default setting). This order sorts items alphabetically with uppercase letters preceding lowercase letters. For example, “A, b, C” would be sorted as “A, C, b.”

Select this box to use international sort order as defined by your operating system. This order is always alphabetic, regardless of case; the letters from the previous example would be sorted as “A, b, C.” See your operating system documentation concerning the sorting of accented characters.

Use Long Qualifiers (Windows only): Select this check box to use long path names as table qualifiers. When you select this check box, path names can be up to 255 characters. The default length for pathnames is 128 characters.

Allow Update and Delete: Specifies whether a data source allows Update and Delete statements. The default is 0. Because Update and Delete statements cause immediate changes to a table, only one connection at a time can operate on a table. When this option is set, tables are opened exclusively by the current connection. Each update and delete on a text file can cause significant changes to the file, and performance may be poor. Consider a more appropriate database form if performance is a significant factor.

Application Using Threads: A setting that ensures that the driver works with multi-threaded applications. You can clear this check box when using the driver with single-threaded applications. Turning off this setting avoids additional processing required for ODBC thread safety standards.

Note

IDL does not support multi-threading.

Define: Click **Define** to define the structure of your text files as described in [“Defining Table Structure”](#) on page 135.

Translate: Click **Translate** to display the **Select Translator** dialog box, which lists the translators specified in the ODBC Translators section of the system information. INTERSOLV provides a translator named INTERSOLV OEM ANSI that translates your data from the IBM PC character set to the ANSI character set.

Select a translator; then, click **OK** to close this dialog box and perform the translation.

On the Macintosh

6. Click the **Mac File Types** tab to specify the creator and file types. Specify 4-character, case-sensitive values for the following:
 - Text File Creator (default is `txtxt`)
 - Text File Type (default is `TEXT`)

Use Macintosh File Types: Use this setting to locate tables based on the Macintosh file type.

Use DOS File Extensions: Use this setting to locate tables based on the DOS file extension.

Include file with matching file types: Select this box to return additional tables based on file type (only if “**Macintosh File Types**” is selected). Enter the file types in the field as a comma-separated list.

Include file with matching extensions: Select this box to return additional tables based on DOS file extension (only if “**DOS File Extensions**” is selected). Enter the extensions in the field as a comma-separated list.

7. On the Macintosh, click the **Define** tab to define the structure of your text files as described in “[Defining Table Structure](#)” on page 135.
8. Click **OK** or **Cancel**. If you click **OK**, the values you have specified become the defaults when you connect to the data source. You can change these defaults by using this procedure to reconfigure your data source. You can override these defaults by connecting to the data source using a connection string with alternate values.

Defining Table Structure

Note

This section does not apply to the UNIX platforms. See [“Defining Table Structure on UNIX Platforms”](#) on page 138 for information on how to define table structure on the UNIX platforms.

Because text files do not all have the same structure, the driver provides the option of defining the structure of an existing file. Although defining the structure is not mandatory (the driver can attempt to guess the names and types of the columns), this feature is extremely useful.

Define the structure of a file as follows:

1. Display the **ODBC Text Driver Setup** dialog box through the ODBC Administrator. Click the **Advanced** tab; then, click **Define** to display the **Define File** dialog box.

Note

On the Macintosh, click the **Define** tab instead of the **Advanced** tab. A pop-up menu appears at the top of the **Define** tab. Use this menu to select a database directory and file. The name of the directory and file are displayed on the tab after you have selected them.

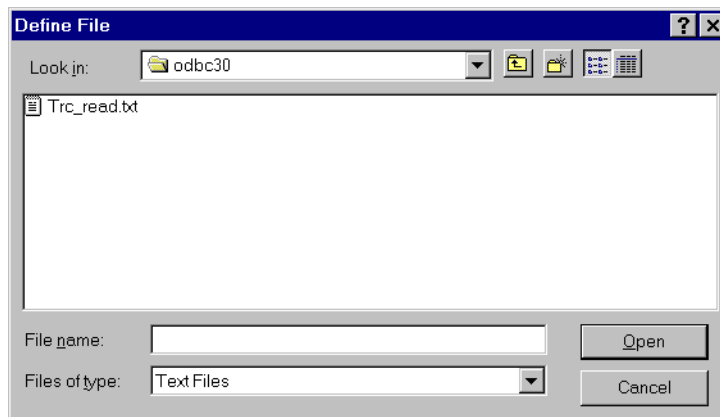
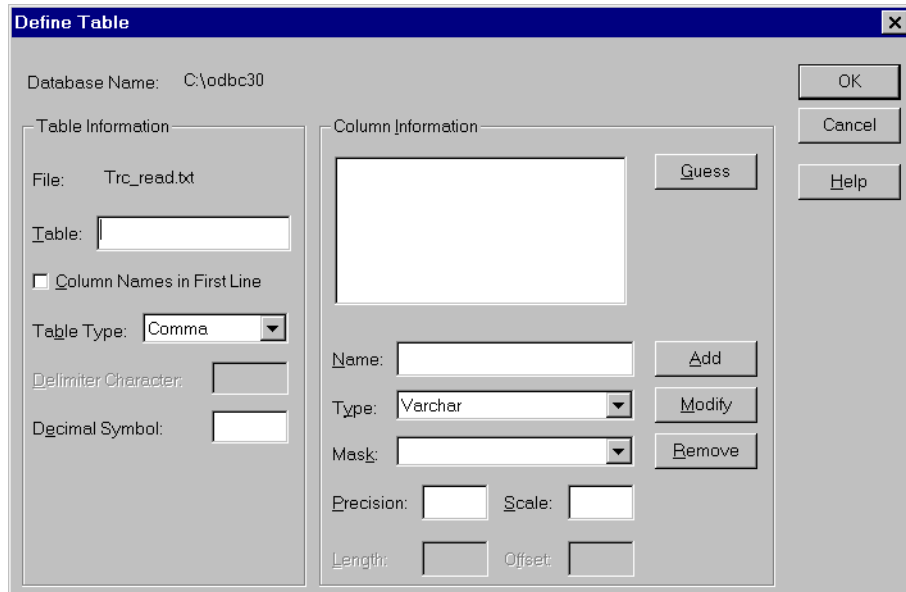


Figure 5-16: The **Define File** dialog box.

2. Select the correct file and click **Open** to display the **Define Table** dialog box.

Note

On the Macintosh, the **Define** tab is equivalent to the **Define Table** dialog box.



*Figure 5-17: The **Define Table** dialog box.*

Database Name: The name of the database directory that you selected in the **Define File** dialog box.

File: The name of the file that you selected in the **Define File** dialog box.

Table: Type a table name in the Table box. The name may be up to 32 characters in length and must be unique. This name is returned by SQLTables. By default, it is the file name without its extension.

Column Names in First Line: Select this check box if the first line of the file contains column names; otherwise, do not select this box.

Table Type: Select either comma, tab, fixed, character, or stream.

Delimiter Character: If the table type is Character, specify the delimiter used in character-separated files.

Decimal Symbol: Type a comma to store the data using a comma as the separator for decimal numbers.

3. If you specified a *comma-separated*, *tab-separated*, or *character-separated* type in the **Table Type** field, the **Guess** button is active and you can click it to have the driver guess at the column names and display them in the list box of the **Column Information** pane.

If you specified a *fixed-length* or *stream* type in the **Table Type** field, the **Parse** button is active and you can click it to display the **Parse Table** dialog box and define the table columns.

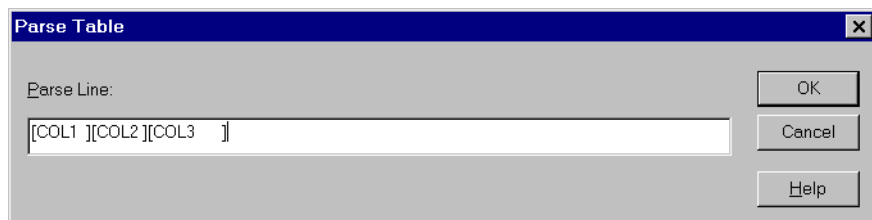


Figure 5-18: The **Parse Table** dialog box.

This dialog box displays the first line of the file. You must mark where *each* field begins and ends by enclosing it in brackets. These brackets indicate the position and length of each field value in the record. Click **OK** to close the **Parse Table** dialog box. The driver will suggest column names in the list box of the **Column Information** pane.

4. If you do not want the driver to guess or parse, enter values in the following fields to define each column. Click **Add** to add the column name to the list box.

Note

On the Macintosh, the **Name**, **Type**, **Mask**, **Precision**, **Scale**, **Length**, and **Offset** fields are displayed on a separate dialog. To define a column, first click **Add**. After setting the values, click **OK** to exit the dialog. The column name will be added to the list box on the **Define** tab.

Name: Type the name of the column.

Type: Select the data type of the column. If the field type is **Date**, you must select a date mask for the field or type one in. See “[Date Masks](#)” on page 140 for more information.

Precision: Type the precision of the column. The precision of numeric data types is defined as the maximum number of digits used by the data type of the column. For character types, this is the length in characters of the data; for binary data types, precision is defined as the length in bytes of the data. For time, timestamp, and all interval data types, precision is the number of characters in the character representation of this data.

Scale: Type the scale of the column. The scale of decimal and numeric data types is defined as the maximum number of digits to the right of the decimal point. For approximate floating point number columns, the scale is undefined, since the number of digits to the right of the decimal point is not fixed. For datetime or interval data that contains a seconds component, the scale is defined as the number of digits to the right of the decimal point in the seconds component of the data.

Note: The precision and scale values determine how numeric data is to be returned.

Length: If you specified a fixed-length table type, length is the number of bytes the data takes up in storage.

Offset: If you specified a fixed-length table type, offset is the number of bytes from the start of the table to the start of the field.

5. To modify an existing column definition, select the column name in the list box. Modify the values for that column name; then, click **Modify**.

Note

On the Macintosh, select the column name; then, click **Modify** to display the separate dialog. Modify the values for that column name; then, click **OK**.

6. To delete an existing column definition, select a column name in the list box and click **Remove**.
7. Click **OK** to define the table.

Defining Table Structure on UNIX Platforms

Because text files do not all have the same structure, the driver provides the option to define the structure of an existing file. Although defining the structure is not

mandatory, because the driver can attempt to guess the names and types of the columns, this feature is extremely useful.

To define the structure of a text file, you create a `QETXT.INI` file using any plain text editor, such as `vi`. The filename must be in uppercase. All of the tables you wish to define are specified in the `QETXT.INI` file. When you specify table attributes in `QETXT.INI`, you override the attributes specified in system information or in the connection string.

Define the `QETXT.INI` file as follows:

1. Create a [Defined Tables] section and list all of the tables you are defining. Specify the text filename (in either upper- or lowercase, depending on the file) followed by the name you want to give the table, for example:

```
emptxt.txt=EMP
```

Table names can be up to 32 characters in length and cannot be the same as another defined table in the database. This name is returned by `SQLTables`. By default, it is the filename without its extension.

2. For each table listed in the [Defined Tables] section, you must specify the text file (`FILE=`), the table type (`TT=`), whether the first line of the file contains column names (`FLN=`), and the delimiter character (`DC=`).

Specify the text filename. For example:

```
FILE=emptxt.txt
```

To define the table type, specify how the fields are separated (comma, tab, fixed, or character). For example:

```
TT=COMMA
```

If the table type is `CHARACTER`, specify the delimiter character. For example, if the fields are separated by semicolons,

```
DC=;
```

Specify whether the first line of the file contains column names, using 1 for yes and 0 for no. For example:

```
FLN=0
```

3. Define the fields in the table, beginning with `FIELD1`. For each field, specify the field name, field type, precision, scale, length, offset (for fixed tables), and date/time mask. See [“Date Masks”](#) on page 140 for information about masks.

Separate the values with commas. For example, to define 2 fields,

```
FIELD1=EMP_ID, VARCHAR, 6, 0, 6, 0,
FIELD2=HIRE_DATE, DATE, 10, 0, 10, 0, m/d/yy
```

4. Save the file as QETXT.INI. The driver looks for this file in the directory specified by the “Database” attribute in odbc.ini, or in the current directory.

Example of QETXT.INI

The following is an example of a QETXT.INI file. This file defines the structure of the emptext.txt file, which is a sample data file shipped with the DataDirect ODBC Text file.

```
[Defined Tables]
emptext.txt=EMP

[EMP]
FILE=emptext.txt
FLN=1
TT=Comma
Charset=ANSI
FIELD1=FIRST_NAME, VARCHAR, 10, 0, 10, 0,
FIELD2=LAST_NAME, VARCHAR, 9, 0, 9, 0,
FIELD3=EMP_ID, VARCHAR, 6, 0, 6, 0,
FIELD4=HIRE_DATE, DATE, 10, 0, 10, 0, m/d/yy
FIELD5=SALARY, NUMERIC, 8, 2, 8, 0,
FIELD6=DEPT, VARCHAR, 4, 0, 4, 0,
FIELD7=EXEMPT, VARCHAR, 6, 0, 6, 0,
FIELD8=INTERESTS, VARCHAR, 136, 0, 136, 0,
```

Date Masks

Date masks tell the driver how a date is stored in a text file. When a value is inserted into a text file, the date is formatted so that it matches the mask. When reading a text file, the driver converts the formatted date into a date data type.

The following table lists the symbols to use when specifying the date mask.

Symbol	Description
m	Output the month's number (1–12).
mm	Output a leading zero if the month number is less than 10.

Table 5-11: : Data Masks for Text Driver

Symbol	Description
mmm, Mmm, MMM	Output the three-letter abbreviation for the month depending on the case of the Ms (that is, jan, Jan, JAN).
mmmm, Mmmm, MMMM	Output the full month name depending on the case of the Ms (that is, january, January, JANUARY).
d	Output the day number (1–31).
dd	Output a leading zero if the day number is less than 10.
ddd, Ddd, DDD	Output the three-letter day abbreviation depending on the case of the Ds (that is, mon, Mon, MON).
dddd, Dddd, DDDD	Output the day depending on the case of the Ds (that is, monday, Monday, MONDAY).
yy	Output the last two digits of the year.
yyyy	Output the full four digits of the year.
J	Output the Julian value for the date. The Julian value is the number of days since 4712 BC.
\ - . : , (space)	Special characters used to separate the parts of a date.
\	Output the next character. For example, if the mask is mm/dd/yyyy \A\D, the value appears as 10/01/1993 AD in the text file.
"string", 'string'	Output the string in the text file.

Table 5-11: : Data Masks for Text Driver

The following table shows some example date values, masks, and how the date appears in the text file.

Date	Mask	Value
1993-10-01	yyyy-mm-dd	1993-10-01
	m/d/yy	10/1/93
	Ddd, Mmm dd, yyyy	Fri, Oct 01, 1993

Table 5-12: Date Mask Examples

Connecting to a Data Source Using a Connection String

If your application requires a connection string to connect to a data source, you must specify the data source name that tells the driver which section in the system information to use for the default connection information. Optionally, you may specify *attribute=value* pairs in the connection string to override the default values stored in the system information. These values are not written to the system information.

You can specify either long or short names in the connection string. The connection string has the form:

```
DSN=data_source_name[;attribute=value
[;attribute=value]...]
```

An example of a connection string for text files is:

```
DSN=TEXT FILES;TT=CHARACTER;DC=&
```

The following table gives the long and short names for each attribute, as well as a description.

To configure a data source in the UNIX environment, you must edit the system information file. This file accepts only long names for attributes. For information about this file, see [“The UNIX Environment”](#) on page 149.

The following table also lists the initial defaults that apply when no value is specified in either the connection string or in the data source definition in the system information. If you specified a value for the attribute when configuring the data source, that value is your default.

Attribute	Description
AllowUpdateAndDelete (AUD)	AllowUpdateAndDelete={0 1}. Specifies whether a data source allows Update and Delete statements. The default is 0. Because Update and Delete statements cause immediate changes to a table, only one connection at a time can operate on a table. When this option is set, tables are opened exclusively by the current connection. Each update and delete on a text file can cause significant changes to the file, and performance may be poor. Consider a more appropriate database form if performance is a significant factor.
ApplicationUsingThreads (AUT)	ApplicationUsingThreads={0 1}. A setting that ensures that the driver works with multi-threaded applications. You can clear this check box when using the driver with single-threaded applications. Turning off this setting avoids additional processing required for ODBC thread safety standards. Note - IDL does not support multi-threading.
CacheSize (CSZ)	The number of 64 KB blocks the driver uses to cache database records. The higher the number of blocks, the better the performance. The maximum number of blocks you can set depends on the system memory available. If the cache size is greater than 0, when browsing backwards, you will not be able to see updates made by other users until you run the Select statement again. The initial default is 4.
Database (DB)	The directory in which the text files are stored.
Note - The ScanRows, TableType, Delimiter, FirstLineNames, and Charset attributes apply to tables that have not been defined. These attributes also determine the characteristics of new tables created with the Create Table statement.	

Table 5-13: : Text Connection String Attributes

Attribute	Description
DataFileExtension (DFE)	<p>Specifies the file extension to use for data files. The default Data File Extension setting is <code>TXT</code>. The Data File Extension setting cannot be greater than three characters. The Data File Extension setting is used for all Create Table statements. Sending a Create Table using an extension other than the Data File Extension setting causes an error.</p> <p>In other SQL statements, such as Select or Insert, users can specify an extension other than the Data File Extension setting. The Data File Extension setting is used when no extension is specified.</p>
DataSourceName (DSN)	A string that identifies a Text data source configuration in the system information. Examples include “Accounting” or “Text Files.”
DecimalSymbol (DS)	DecimalSymbol={ . , }. Specifies the decimal separator used when data is stored. The international decimal symbol (.) must be used in DML statements and parameter buffers.
Delimiter (DC)	The character used as a delimiter for character-separated files. It can be any printable character except single or double quotes. The initial default is a comma (,).
ExtraExtensions (EE)	A list of additional filename extensions to be recognized as text tables. When an application requests a list of tables, only files that have been defined are returned. To have the driver also return names of undefined files, specify a comma-separated list of file extensions. To specify files with no extension, use the keyword <code>None</code> .
<p>Note - The ScanRows, TableType, Delimiter, FirstLineNames, and Charset attributes apply to tables that have not been defined. These attributes also determine the characteristics of new tables created with the Create Table statement.</p>	

Table 5-13: : Text Connection String Attributes

Attribute	Description
FileOpenCache (FOC)	The maximum number of unused file opens to cache. For example, when FileOpenCache=4, and a user opens and closes four files, the files are not actually closed. The driver keeps them open so that if another query uses one of these files, the driver does not have to perform another open, which is expensive. The advantage of using file open caching is increased performance. The disadvantage is that a user who tries to open the file exclusively may get a locking conflict even though no one appears to have the file open. The initial default is 0.
FirstLineNames (FLN)	FirstLineNames={0 1}. This attribute determines whether the driver looks for column names in the first line of the file. If FirstLineNames=1, the driver looks for column names in the first line of the file. If FirstLineNames=0 (the initial default), the first line is interpreted as the first record in the file.
IntlSort (IS)	IntlSort={0 1}. This attribute determines the order that records are retrieved when you issue a Select statement with an Order By clause. If IntlSort=0 (the initial default), the driver uses the ASCII sort order. This order sorts items alphabetically with uppercase letters preceding lowercase letters. For example, "A, b, C" would be sorted as "A, C, b." If IntlSort=1, the driver uses the international sort order as defined by your operating system. This order is always alphabetic, regardless of case; the letters from the previous example would be sorted as "A, b, C." See your operating system documentation concerning the sorting of accented characters.
<p>Note - The ScanRows, TableType, Delimiter, FirstLineNames, and Charset attributes apply to tables that have not been defined. These attributes also determine the characteristics of new tables created with the Create Table statement.</p>	

Table 5-13: : Text Connection String Attributes

Attribute	Description
MacCompatible (MC)	MacCompatible={0 1}. On Macintosh systems, allows the user to specify whether tables should be accessed by DOS file extension (0) or Macintosh file type (1). The default is 0. If you are accessing tables by extension and want to return additional tables, use ExtraExtensions. If you are accessing tables by file type and want to return additional tables, use MacFileTypesList.
MacFileInfo (MFI)	On Macintosh systems, four-character, case-sensitive values that specify the following in the order shown: <ul style="list-style-type: none"> • Text File Creator (default is <code>txt</code>) • Text File Type (default is <code>TEXT</code>) The values are specified in a comma-separated list. For example, MacFileInfo=ABCD,EFGH.
MacFileTypesList (MFTL)	On Macintosh systems, specifies which file types will be used when returning additional tables. The default is none. The values are specified in a comma-separated list. For example, MacFileTypesList=ABCD,EFGH,IJKL,MNOP,QRST.
ScanRows (SR)	The number of rows in a text file that the driver scans to determine the column types in the file. If the value is 0, all rows in the file are scanned. The initial default is 25.
TableType (TT)	TableType={Comma Tab Character Fixed Stream}. The Text driver supports four types: comma-separated, tab-separated, character-separated, fixed length, and stream. Setting this value tells the driver the default type, which is used when creating a new table and opening an undefined table.
Note - The ScanRows, TableType, Delimiter, FirstLineNames, and Charset attributes apply to tables that have not been defined. These attributes also determine the characteristics of new tables created with the Create Table statement.	

Table 5-13: : Text Connection String Attributes

Attribute	Description
UndefinedTable (UT)	The Text driver can perform two operations when it encounters a file that has not been defined. UndefinedTable=Prompt tells the driver to display a dialog box that allows the user to describe the file's format. UndefinedTable=Guess tells the driver to guess the file's format. This is the initial default.
UseLongQualifiers (ULQ) (Windows only)	UseLongQualifiers={0 1}. It specifies whether the driver uses long pathnames as table qualifiers. The default is 0, do not use long path names (the default length of path names is 128 characters). If UseLongQualifiers=1, the driver uses long path names (up to 255 characters).
Note - The ScanRows, TableType, Delimiter, FirstLineNames, and Charset attributes apply to tables that have not been defined. These attributes also determine the characteristics of new tables created with the Create Table statement.	

Table 5-13: : Text Connection String Attributes

Data Types

The following table shows how the text file data types are mapped to the standard ODBC data types.

Text	ODBC
Numeric	SQL_NUMERIC
Date	SQL_TYPE_DATE
Varchar	SQL_VARCHAR

Table 5-14: Text Data Types

Select Statement

You use the SQL Select statement to specify the columns and records to be read.

Alter Table Statement

The Text driver supports the Alter Table statement to add one or more columns to a table or to delete (drop) a single column.

The Alter Table statement has the form:

```
ALTER TABLE table_name
{ADD column_name data_type
| ADD (column_name data_type
[ ,column_name data_type] ...)
| DROP [COLUMN] column_name}
```

table_name is the name of the table to which you are adding or dropping columns.

column_name assigns a name to the column you are adding or specifies the column you are dropping.

data_type specifies the native data type of each column you add.

For example, to add two columns to the emp table,

```
ALTER TABLE emp (ADD startdate date, dept varchar(10))
```

You cannot add columns and drop columns in a single statement, and you can drop only one column at a time. For example, to drop a column,

```
ALTER TABLE emp DROP startdate
```

The Alter Table statement fails when you attempt to drop a column upon which other objects, such as indexes or views, are dependent.

ODBC Conformance Level

The Text driver supports the functions listed in [Chapter 6, “ODBC API and Scalar Functions”](#). In addition, the following function is supported: SQLSetPos.

The Text driver also supports backward and random fetching in SQLExtendedFetch and SQLFetchScroll. The driver supports the minimum SQL grammar.

Number of Connections and Statements Supported

Text files support multiple connections and multiple statements per connection.

The UNIX Environment

This section contains specific information about using Connect ODBC in the UNIX environment.

The System Information File (.odbc.ini)

In the UNIX environment, there is no ODBC Administrator. To configure a data source, you must edit the system information file, a plain text file that is normally located in the user's \$HOME directory and is usually called `.odbc.ini`. This file is maintained using any text editor to define data source entries as described in the "Connecting to a Data Source Using a Connection String" section of each driver's chapter. A sample file (`odbc.ini`) is located in the driver installation directory.

UNIX support of the database drivers also permits the use of a centralized system information file that a system administrator can control. This is accomplished by setting the environment variable `ODBCINI` to point to the fully qualified pathname of the centralized file. For example, in the C shell you could set this variable as follows:

```
setenv ODBCINI /opt/odbc/system_odbc.ini
```

In the Bourne or Korn shell, you would set it as:

```
ODBCINI=/opt/odbc/system_odbc.ini;export ODBCINI
```

The search order for the location of the system information file is as follows:

1. Check `ODBCINI`
2. Check `$HOME` for `.odbc.ini`

There must be an `[ODBC]` section in the system information file that includes the `InstallDir` keyword. The value of this keyword must be the path to the directory under which the `/lib` and `/messages` directories are contained. For example, if you choose the default install directory, then the following line must be in the `[ODBC]` section:

```
InstallDir=/opt/odbc
```

Sample Solaris System Information File

In the following sample, `xx` represents the driver number:

```
[ODBC Data Sources]
Oracle7=Sample Oracle dsn
dBase=Sample dBASE dsn
Sybase=Sample Sybase dsn
```

```
Informix=Sample Infofrmix dsn
OpenIngres=Sample OpenIngres dsn
DB2=Sample DB2 dsn
Text=Sample Text file dsn
[dBase]
Driver=/opt/odbc/lib/ivdbfxx.so
Description=dBase
Database=/opt/odbc/demo
[Sybase]
Driver=/opt/odbc/lib/ivsybxx.so
Description=Sybase
Database=odbc
ServerName=SYBASE
LogonID=odbc01
Password=odbc01
OptimizePrepare=2
SelectMethod=1
[Oracle7]
Driver=/opt/odbc/lib/ivor7xx.so
Description=Oracle7
ServerName=oraclehost
LogonID=odbc01
Password=odbc01
[Informix]
Driver=/opt/odbc/lib/ivinfxx.so
Description=Informix7
Database=odbc
HostName=informixhost
LogonID=odbc01
Password=odbc01
[DB2]
Driver=/opt/odbc/lib/ivdb2xx.so
Description=DB2
Database=ODBC
[OpenIngres]
Driver=/opt/odbc/lib/ivoingxx.so
ServerName=ingreshost
Database=odbc
LogonID=odbc01
Password=odbc01
[Text]
Driver=/opt/odbc/lib/ivtxtxx.so
Description=Text driver
Database=/opt/odbc/demo
[ODBC]
Trace=0
TraceFile=odbctrace.out
TraceDll=/opt/odbc/lib/odbctrac.so
InstallDir=/opt/odbc
```

Optional Environment Variables

Many of the Connect ODBC drivers must have environment variables set as required by the database client components used by the drivers. Consult the system requirements in each of the individual driver sections for additional information pertaining to individual driver requirements.

ODBCINI is an optional environment variable that all Connect ODBC drivers will recognize. ODBCINI is used to locate a system information file other than the default file and is described in detail under “[The System Information File \(.odbc.ini\)](#)” on page 149.

Using Double-Byte Character Sets

Connect ODBC drivers are capable of using double-byte character sets. The drivers normally use the character set defined by the default locale “C” unless explicitly pointed to another character set. The default locale “C” corresponds to the 7-bit ASCII character set in which only characters from ISO 8859-1 are valid.

Note

For more information, see the man pages for “locale” and “setlocale.”

Translators

INTERSOLV provides a sample translator named INTERSOLV OEM ANSI that translates provides a framework for coding a translation library.

You must add the TranslationSharedLibrary keyword to the data source section of the system information file to perform a translation. Adding the TranslationOption keyword is optional.

Keyword	Definition
TranslationSharedLibrary	Full path of translation library.
TranslationOption	ASCII representation of the 32-bit integer translation option.

Table 5-15: : TranslationSharedLibrary Keyword

Locking and Isolation Levels

This section discusses locking and isolation levels and how their settings can affect the data you retrieve. Different database systems support different locking and isolation levels. See the section “Isolation and Lock Levels Supported” in the appropriate driver chapter.

Locking

Locking is a database operation that restricts a user from accessing a table or record. Locking is used in situations where more than one user might try to use the same table or record at the same time. By locking the table or record, the system ensures that only one user at a time can affect the data.

Locking is critical in multiuser databases, where different users can try to access or modify the same records concurrently. Although such concurrent database activity is desirable, it can create problems. Without locking, for example, if two users try to modify the same record at the same time, they might encounter problems ranging from retrieving bad data to deleting data that the other user needs. If, however, the first user to access a record can lock that record to temporarily prevent other users from modifying it, such problems can be avoided. Locking provides a way to manage concurrent database access while minimizing the various problems it can cause.

Isolation Levels

An isolation level represents a particular locking strategy employed in the database system to improve data consistency. The higher the isolation level, the more complex the locking strategy behind it. The isolation level provided by the database determines whether a transaction will encounter the following behaviors in data consistency:

Behavior	How it Appears
Dirty reads	User 1 modifies a row. User 2 reads the same row before User 1 commits. User 1 performs a rollback. User 2 has read a row that has never really existed in the database. User 2 may base decisions on false data.

Table 5-16: Database Inconsistencies Due to Isolation Level

Behavior	How it Appears
Non-repeatable reads	User 1 reads a row but does not commit. User 2 modifies or deletes the same row and then commits. User 1 rereads the row and finds it has changed (or has been deleted).
Phantom reads	User 1 uses a search condition to read a set of rows but does not commit. User 2 inserts one or more rows that satisfy this search condition, then commits. User 1 rereads the rows using the search condition and discovers rows that were not present before.

Table 5-16: Database Inconsistencies Due to Isolation Level

Isolation levels represent the database system's ability to prevent these behaviors. The American National Standards Institute (ANSI) defines four isolation levels:

- Read uncommitted (0)
- Read committed (1)
- Repeatable read (2)
- Serializable (3)

In ascending order (0–3), these isolation levels provide an increasing amount of data consistency to the transaction. At the lowest level, all three behaviors can occur. At the highest level, none can occur. The success of each level in preventing these behaviors is due to the locking strategies that they employ, which are as follows:

ANSI Isolation Levels	Resulting Locking Strategy
Read uncommitted (0)	Locks are obtained on modifications to the database and held until end of transaction (EOT). Reading from the database does not involve any locking.
Read committed (1)	Locks are acquired for reading and modifying the database. Locks are released after reading but locks on modified objects are held until EOT.

Table 5-17: ANSI Isolation Levels and Resulting Locking Strategy

ANSI Isolation Levels	Resulting Locking Strategy
Repeatable read (2)	Locks are obtained for reading and modifying the database. Locks on all modified objects are held until EOT. Locks obtained for reading data are held until EOT. Locks on non-modified access structures (such as indexes and hashing structures) are released after reading.
Serializable (3)	All data read or modified is locked until EOT. All access structures that are modified are locked until EOT. Access structures used by the query are locked until EOT.

Table 5-17: ANSI Isolation Levels and Resulting Locking Strategy

Table 5-16 shows what data consistency behaviors can occur at each isolation level.

Level	Dirty Read	Nonrepeatable Read	Phantom Read
0, Read uncommitted	Yes	Yes	Yes
1, Read committed	No	Yes	Yes
2, Repeatable read	No	No	Yes
3, Serializable	No	No	No

Table 5-18: Isolation Levels and Data Consistency

Although higher isolation levels provide better data consistency, this consistency can be costly in terms of the concurrency provided to individual users. Concurrency is the ability of multiple users to access and modify data simultaneously. As isolation levels increase, so does the chance that the locking strategy used will create problems in concurrency.

Put another way: The higher the isolation level, the more locking involved, and the more time users may spend waiting for data to be freed by another user. Because of this inverse relationship between isolation levels and concurrency, you must consider how people use the database before choosing an isolation level. You must weigh the trade-offs between data consistency and concurrency, and decide which is more important.

Locking Modes and Levels

Different database systems employ various locking modes, but they have two basic ones in common: shared and exclusive. Shared locks can be held on a single object by multiple users. If one user has a shared lock on a record, then a second user can also get a shared lock on that same record; however, the second user cannot get an exclusive lock on that record. Exclusive locks are exclusive to the user that obtains them. If one user has an exclusive lock on a record, then a second user cannot get either type of lock on the same record.

Performance and concurrency can also be affected by the locking level used in the database system. The locking level determines the size of an object that is locked in a database. For example, many database systems let you lock an entire table, as well as individual records. An intermediate level of locking, page-level locking, is also common. A page contains one or more records and is typically the amount of data read from the disk in a single disk access. The major disadvantage of page-level locking is that if one user locks a record, a second user may not be able to lock other records because they are stored on the same page as the locked record.



Chapter 6:

ODBC API and Scalar Functions

This chapter lists the ODBC API functions that the ODBC drivers support and the scalar functions, which you use in SQL statements.

API Functions	158	Scalar Functions	161
-------------------------------------	-----	--	-----

API Functions

All database drivers are ODBC Level 1–compliant—they support all ODBC Core and Level 1 functions. A limited set of Level 2 functions is also supported. The drivers support the functions listed in the following tables. Any additions to these supported functions or differences in the support of specific functions are listed in the “ODBC Conformance Level” section in the individual driver chapters.

Core Functions	Level 1 Functions
SQLAllocConnect	SQLColumns
SQLAllocEnv	SQLDriverConnect
SQLAllocStmt	SQLGetConnectOption
SQLBindCol	SQLGetData
SQLBindParameter	SQLGetFunctions
SQLCancel	SQLGetInfo
SQLColAttributes	SQLGetStmtOption
SQLConnect	SQLGetTypeInfo
SQLDescribeCol	SQLParamData
SQLDisconnect	SQLPutData
SQLDrivers	SQLSetConnectOption
SQLError	SQLSetStmtOption
SQLExecDirect	SQLSpecialColumns
SQLExecute	SQLStatistics
SQLFetch	SQLTables
SQLFreeConnect	Level 2 Functions
SQLFreeEnv	SQLBrowseConnect (all drivers except PROGRESS)
SQLFreeStmt	SQLDataSources
SQLGetCursorName	SQLExtendedFetch (forward scrolling only)

Core Functions**Level 1 Functions**

SQLNumResultCols	SQLMoreResults
SQLPrepare	SQLNativeSql
SQLRowCount	SQLNumParams
SQLSetCursorName	SQLParamOptions
SQLTransact	SQLSetScrollOptions
SQLAllocHandle	SQLGetData
SQLBindCol	SQLGetDescField
SQLBindParameter	SQLGetDescRec
SQLBrowseConnect (except for PROGRESS)	SQLGetDiagField
SQLBulkOperations	SQLGetDiagRec
SQLCancel	SQLGetEnvAttr
SQLCloseCursor	SQLGetFunctions
SQLColAttribute	SQLGetInfo
SQLColumns	SQLGetStmtAttr
SQLConnect	SQLGetTypeInfo
SQLCopyDesc	SQLMoreResults
SQLDataSources	SQLNativeSql
SQLDescribeCol	SQLNumParens
SQLDisconnect	SQLNumResultCols
SQLDriverConnect	SQLParamData
SQLDrivers	SQLPrepare
SQLEndTran	SQLPutData
SQLError	SQLRowCount
SQLExecDirect	SQLSetConnectAttr
SQLExecute	SQLSetCursorName
SQLExtendedFetch	SQLSetDescField

Core Functions

SQLFetch
SQLFetchScroll (forward scrolling only)
SQLFreeHandle
SQLFreeStmt
SQLGetConnectAttr
SQLGetCursorName

Level 1 Functions

SQLSetDescRec
SQLSetEnvAttr
SQLSetStmtAttr
SQLSpecialColumns
SQLStatistics
SQLTables

Scalar Functions

The following tables list the scalar functions that ODBC supports; your database system may not support all of these functions. See the documentation for your database system to find out which functions are supported.

You can use these functions in SQL statements using the following syntax:

```
{fn scalar-function}
```

where *scalar-function* is one of the functions listed in the following tables. For example,

```
SELECT {fn UCASE(NAME)} FROM EMP
```

String Functions

The following table lists the string functions that ODBC supports.

The string functions listed can take the following arguments:

- *string_exp* can be the name of a column, a string literal, or the result of another scalar function, where the underlying data type is SQL_CHAR, SQL_VARCHAR, or SQL_LONGVARCHAR.
- *start*, *length*, and *count* can be the result of another scalar function or a literal numeric value, where the underlying data type is SQL_TINYINT, SQL_SMALLINT, or SQL_INTEGER.

The string functions are one-based; that is, the first character in the string is character 1.

Character string literals must be surrounded in single quotation marks.

Function	Returns
ASCII(<i>string_exp</i>)	ASCII code value of the leftmost character of <i>string_exp</i> as an integer.
BIT_LENGTH(<i>string_exp</i>) ODBC 3.0	The length in bits of the string expression.

Table 6-1: Scalar String Functions

Function	Returns
CHAR(<i>code</i>)	The character with the ASCII code value specified by <i>code</i> . <i>code</i> should be between 0 and 255; otherwise, the return value is data-source dependent.
CHAR_LENGTH(<i>string_exp</i>) ODBC 3.0	The length in characters of the string expression, if the string expression is of a character data type; otherwise, the length in bytes of the string expression (the smallest integer not less than the number of bits divided by 8). (This function is the same as the CHARACTER_LENGTH function.)
CHARACTER_LENGTH(<i>string_exp</i>) ODBC 3.0	The length in characters of the string expression, if the string expression is of a character data type; otherwise, the length in bytes of the string expression (the smallest integer not less than the number of bits divided by 8). (This function is the same as the CHAR_LENGTH function.)
CONCAT(<i>string_exp1</i> , <i>string_exp2</i>)	The string resulting from concatenating <i>string_exp2</i> and <i>string_exp1</i> . The string is system dependent.
DIFFERENCE(<i>string_exp1</i> , <i>string_exp2</i>)	An integer value that indicates the difference between the values returned by the SOUNDEX function for <i>string_exp1</i> and <i>string_exp2</i> .
INSERT(<i>string_exp1</i> , <i>start</i> , <i>length</i> , <i>string_exp2</i>)	A string where <i>length</i> characters have been deleted from <i>string_exp1</i> beginning at <i>start</i> and where <i>string_exp2</i> has been inserted into <i>string_exp</i> , beginning at <i>start</i> .
LCASE(<i>string_exp</i>)	Uppercase characters in <i>string_exp</i> converted to lowercase.
LEFT(<i>string_exp</i> , <i>count</i>)	The <i>count</i> of characters of <i>string_exp</i> .

Table 6-1: Scalar String Functions (Continued)

Function	Returns
LENGTH(<i>string_exp</i>)	The number of characters in <i>string_exp</i> , excluding trailing blanks and the string termination character.
LOCATE(<i>string_exp1</i> , <i>string_exp2</i> [, <i>start</i>])	The starting position of the first occurrence of <i>string_exp1</i> within <i>string_exp2</i> . If <i>start</i> is not specified the search begins with the first character position in <i>string_exp2</i> . If <i>start</i> is specified, the search begins with the character position indicated by the value of <i>start</i> . The first character position in <i>string_exp2</i> is indicated by the value 1. If <i>string_exp1</i> is not found, 0 is returned.
LTRIM(<i>string_exp</i>)	The characters of <i>string_exp</i> , with leading blanks removed.
OCTET_LENGTH(<i>string_exp</i>) ODBC 3.0	The length in bytes of the string expression. The result is the smallest integer not less than the number of bits divided by 8.
POSITION(<i>character_exp</i> IN <i>character_exp</i>)ODBC 3.0	The position of the first character expression in the second character expression. The result is an exact numeric with an implementation-defined precision and a scale of 0.
REPEAT(<i>string_exp</i> , <i>count</i>)	A string composed of <i>string_exp</i> repeated <i>count</i> times.
REPLACE(<i>string_exp1</i> , <i>string_exp2</i> , <i>string_exp3</i>)	Replaces all occurrences of <i>string_exp2</i> in <i>string_exp1</i> with <i>string_exp3</i> .
RIGHT(<i>string_exp</i> , <i>count</i>)	The rightmost <i>count</i> of characters in <i>string_exp</i> .
RTRIM(<i>string_exp</i>)	The characters of <i>string_exp</i> with trailing blanks removed.
SOUNDEX(<i>string_exp</i>)	A data-source-dependent string representing the sound of the words in <i>string_exp</i> .

Table 6-1: Scalar String Functions (Continued)

Function	Returns
SPACE(<i>count</i>)	A string consisting of <i>count</i> spaces.
SUBSTRING(<i>string_exp</i> , <i>start</i> , <i>length</i>)	A string derived from <i>string_exp</i> beginning at the character position <i>start</i> for <i>length</i> characters.
UCASE(<i>string_exp</i>)	Lowercase characters in <i>string_exp</i> converted to uppercase.

Table 6-1: Scalar String Functions (Continued)

Numeric Functions

The following table lists the numeric functions that ODBC supports.

The numeric functions listed can take the following arguments:

- *numeric_exp* can be a column name, a numeric literal, or the result of another scalar function, where the underlying data type is SQL_NUMERIC, SQL_DECIMAL, SQL_TINYINT, SQL_SMALLINT, SQL_INTEGER, SQL_BIGINT, SQL_FLOAT, SQL_REAL, or SQL_DOUBLE.
- *float_exp* can be a column name, a numeric literal, or the result of another scalar function, where the underlying data type is SQL_FLOAT.
- *integer_exp* can be a column name, a numeric literal, or the result of another scalar function, where the underlying data type is SQL_TINYINT, SQL_SMALLINT, SQL_INTEGER, or SQL_BIGINT.

Function	Returns
ABS(<i>numeric_exp</i>)	Absolute value of <i>numeric_exp</i> .
ACOS(<i>float_exp</i>)	Arccosine of <i>float_exp</i> as an angle in radians.
ASIN(<i>float_exp</i>)	Arcsine of <i>float_exp</i> as an angle in radians.
ATAN(<i>float_exp</i>)	Arctangent of <i>float_exp</i> as an angle in radians.
ATAN2(<i>float_exp1</i> , <i>float_exp2</i>)	Arctangent of the <i>x</i> - and <i>y</i> -coordinates, specified by <i>float_exp1</i> and <i>float_exp2</i> as an angle in radians.

Table 6-2: Scalar Numeric Function

Function	Returns
CEILING(<i>numeric_exp</i>)	Smallest integer greater than or equal to <i>numeric_exp</i> .
COS(<i>float_exp</i>)	Cosine of <i>float_exp</i> as an angle in radians.
COT(<i>float_exp</i>)	Cotangent of <i>float_exp</i> as an angle in radians.
DEGREES(<i>numeric_exp</i>)	Number if degrees converted from <i>numeric_exp</i> radians.
EXP(<i>float_exp</i>)	Exponential value of <i>float_exp</i> .
FLOOR(<i>numeric_exp</i>)	Largest integer less than or equal to <i>numeric_exp</i> .
LOG(<i>float_exp</i>)	Natural log of <i>float_exp</i> .
LOG10(<i>float_exp</i>)	Base 10 log of <i>float_exp</i> .
MOD(<i>integer_exp1</i> , <i>integer_exp2</i>)	Remainder of <i>integer_exp1</i> divided by <i>integer_exp2</i> .
PI()	Constant value of pi as a floating-point number.
POWER(<i>numeric_exp</i> , <i>integer_exp</i>)	Value of <i>numeric_exp</i> to the power of <i>integer_exp</i> .
RADIANS(<i>numeric_exp</i>)	Number of radians converted from <i>numeric_exp</i> degrees.
RAND([<i>integer_exp</i>])	Random floating-point value using <i>integer_exp</i> as the optional seed value.
ROUND(<i>numeric_exp</i> , <i>integer_exp</i>)	<i>numeric_exp</i> rounded to <i>integer_exp</i> places right of the decimal (left of the decimal if <i>integer_exp</i> is negative).
SIGN(<i>numeric_exp</i>)	Indicator of the sign of <i>numeric_exp</i> . If <i>numeric_exp</i> < 0, -1 is returned. If <i>numeric_exp</i> = 0, 0 is returned. If <i>numeric_exp</i> > 0, 1 is returned.
SIN(<i>float_exp</i>)	Sine of <i>float_exp</i> , where <i>float_exp</i> is an angle in radians.

Table 6-2: Scalar Numeric Function

Function	Returns
SQRT(<i>float_exp</i>)	Square root of <i>float_exp</i> .
TAN(<i>float_exp</i>)	Tangent of <i>float_exp</i> , where <i>float_exp</i> is an angle in radians.
TRUNCATE(<i>numeric_exp</i> , <i>integer_exp</i>)	<i>numeric_exp</i> truncated to <i>integer_exp</i> places right of the decimal. (If <i>integer_exp</i> is negative, truncation is to the left of the decimal.)

Table 6-2: Scalar Numeric Function

Date and Time Functions

The following table lists the date and time functions that ODBC supports.

The date and time functions listed can take the following arguments:

- *date_exp* can be a column name, a date or timestamp literal, or the result of another scalar function, where the underlying data type can be represented as SQL_CHAR, SQL_VARCHAR, SQL_DATE, or SQL_TIMESTAMP.
- *time_exp* can be a column name, a timestamp or timestamp literal, or the result of another scalar function, where the underlying data type can be represented as SQL_CHAR, SQL_VARCHAR, SQL_TIME, or SQL_TIMESTAMP.
- *timestamp_exp* can be a column name; a time, date, or timestamp literal; or the result of another scalar function, where the underlying data type can be represented as SQL_CHAR, SQL_VARCHAR, SQL_TIME, SQL_DATE, or SQL_TIMESTAMP.

Function	Returns
CURRENT_DATE() ODBC 3.0	Current date.
CURRENT_TIME[(<i>time-precision</i>)] ODBC 3.0	Current local time. The <i>time-precision</i> argument determines the seconds precision of the returned value.

Table 6-3: Scalar Time and Date Functions

Function	Returns
CURRENT_TIMESTAMP[(<i>timestamp-precision</i>)] ODBC 3.0	Current local date and local time as a timestamp value. The <i>timestamp-precision</i> argument determines the seconds precision of the returned timestamp.
CURDATE()	Current date as a date value.
CURTIME()	Current local time as a time value.
DAYNAME(<i>date_exp</i>)	Character string containing a data-source-specific name of the day for the day portion of <i>date_exp</i> .
DAYOFMONTH(<i>date_exp</i>)	Day of the month in <i>date_exp</i> as an integer value (1–31).
DAYOFWEEK(<i>date_exp</i>)	Day of the week in <i>date_exp</i> as an integer value (1–7).
DAYOFYEAR(<i>date_exp</i>)	Day of the year in <i>date_exp</i> as an integer value (1–366).
HOUR(<i>time_exp</i>)	Hour in <i>time_exp</i> as an integer value (0–23).
MINUTE(<i>time_exp</i>)	Minute in <i>time_exp</i> as an integer value (0–59).
MONTH(<i>date_exp</i>)	Month in <i>date_exp</i> as an integer value (1–12).
MONTHNAME(<i>date_exp</i>)	Character string containing the data source-specific name of the month.
NOW()	Current date and time as a timestamp value.
QUARTER(<i>date_exp</i>)	Quarter in <i>date_exp</i> as an integer value (1–4).
SECOND(<i>time_exp</i>)	Second in <i>date_exp</i> as an integer value (0–59).

Table 6-3: Scalar Time and Date Functions (Continued)

Function	Returns
TIMESTAMPADD(<i>interval</i> , <i>integer_exp</i> , <i>time_exp</i>)	Timestamp calculated by adding <i>integer_exp</i> intervals of type <i>interval</i> to <i>time_exp</i> . <i>interval</i> can be SQL_TSI_FRAC_SECOND SQL_TSI_SECOND SQL_TSI_MINUTE SQL_TSI_HOUR SQL_TSI_DAY SQL_TSI_WEEK SQL_TSI_MONTH SQL_TSI_QUARTER SQL_TSI_YEAR Fractional seconds are expressed in billionths of a second.
TIMESTAMPDIFF(<i>interval</i> , <i>time_exp1</i> , <i>time_exp2</i>)	Integer number of intervals of type <i>interval</i> by which <i>time_exp2</i> is greater than <i>time_exp1</i> . <i>interval</i> has the same values as TIMESTAMPADD. Fractional seconds are expressed in billionths of a second.
WEEK(<i>date_exp</i>)	Week of the year in <i>date_exp</i> as an integer value (1–53).
YEAR(<i>date_exp</i>)	Year in <i>date_exp</i> . The range is data-source dependent.

Table 6-3: Scalar Time and Date Functions (Continued)

System Functions

The following table lists the system functions that ODBC supports.

Function	Returns
DATABASE()	Name of the database, corresponding to the connection handle (<i>hdbc</i>).
IFNULL(<i>exp,value</i>)	<i>value</i> , if <i>exp</i> is null.
USER()	Authorization name of the user.

Table 6-4: Scalar System Functions



Index

Symbols

slash character, [44](#)

A

ABSOLUTE keyword, [66](#)

accessing

data in a database, [10](#)

databases using IDL objects, [24](#)

external databases, [17](#)

Add Data Source dialog, [29](#)

Alter Table statement

Text, [148](#)

API

conformance standard, [12](#)

functions, [12](#)

arguments

keywords, [43](#)

positional parameters, [43](#)

C

calling sequence. *See* syntax

CAN_GET_TABLES keyword, [51](#)

CAN_MOVE_ABSOLUTE keyword, [58](#), [62](#),
[68](#)

CAN_MOVE_FIRST keyword, [62](#)

CAN_MOVE_LAST keyword, [62](#)

CAN_MOVE_NEXT keyword, [62](#)

CAN_MOVE_PRIOR keyword, [62](#)

CAN_MOVE_RELATIVE keyword, [62](#)

conformance

API levels, [12](#)

- Core Level API, [12](#)
- Level 1 API, [12](#)
- Level 2 API, [12](#)
- ODBC standards, [12](#)
- SQL levels, [12](#)
- Connect ODBC
 - INFORMIX, [79](#)
 - Oracle, [93](#)
 - Sybase, [109](#)
 - Text, [127](#)
- CONNECTION keyword, [48](#)
- connections supported
 - INFORMIX, [92](#)
 - Oracle, [108](#)
 - Sybase, [126](#)
 - Text, [148](#)
- conventions
 - terminology, [16](#)
 - used in this manual, [16](#)
- converting data types, [37](#)
- Core Level API conformance, [12](#)

D

- data
 - retrieving from a table, [31](#)
- Data Manipulation Language, [13](#)
- data source, [11](#)
 - configuring
 - INFORMIX, [81](#)
 - Oracle, [95](#)
 - Sybase, [110](#)
 - Text, [128](#)
 - connecting via connection string
 - INFORMIX, [86](#)
 - Oracle, [100](#)
 - Sybase, [119](#)
 - Text, [142](#)
 - connecting via logon dialog box
 - INFORMIX, [85](#)
 - Oracle, [100](#)
 - Sybase, [118](#)
- data types
 - converting, [37](#)
 - INFORMIX, [89](#)
 - Oracle, [106](#)
 - Sybase, [124](#)
 - Text, [147](#)
- database
 - availability
 - DB_Exists function, [25](#)
 - finding a specific database, [26](#)
 - GetDatasources method, [26](#)
 - connecting, [27](#), [47](#)
 - creating an object, [26](#)
 - database application, [11](#)
 - database management systems, [10](#)
 - data source, [11](#)
 - DataMiner Manual
 - Overview, [9](#)
 - DATASOURCE keyword, [48](#)
 - date and time functions, [166](#)
 - date format, [37](#)
 - DB_Exists function, [25](#), [46](#)
 - DBMS, [10](#)
 - DBMS_NAME keyword, [51](#)
 - DBMS_VERSION keyword, [51](#)
 - default data source specification, [72](#)
 - dialog boxes
 - Add Data Source, [29](#)
 - SQL Data Sources, [27](#)
 - DIALOG_DBConnect function, [27](#), [45](#)
 - Driver Manager
 - DriverSet component, [11](#)
 - ODBC architecture, [11](#)
 - DRIVER_ODBC_LEVEL keyword, [51](#)
 - DRIVER_VERSION keyword, [51](#)
 - drivers, [11](#)
 - DriverSet
 - components, [11](#)

E

- error message
 - ODBC formats, 21
 - standard, 21
 - verbose, 21

F

- FIELD_INFO keyword, 63
- files
 - ODBC.INI, 70
 - sql.log, 73
 - trace, 72
- FIRST keyword, 66
- formal parameters, 43
- formats
 - date, 37
 - time, 37
 - timestamp, 37
- formats, for text files, 127
- function calling sequence, 42
- function method calling sequence, 43
- functions
 - data conversion, 37
 - scalar, 37

G

- GET_DATABASE keyword, 63

I

- IDLdbDatabase
 - Connect method, 48
 - ExecuteSQL method, 49
 - GetDatasources method, 50
 - GetProperty method, 51
 - GetTables method, 53
 - SetProperty method, 54

- IDLdbDatabase object, 47
- IDLdbRecordset
 - AddRecord method, 58
 - CurrentRow method, 59
 - DeleteRecord method, 60
 - GetField method, 61
 - GetProperty method, 62
 - IsReadOnly method, 66
 - MoveCursor method, 66
 - nFields method, 67
- IDLdbRecordset
 - GetRecord Method, 65
- IDLdbRecordset object, 56
- INFORMIX driver
 - connections supported, 92
 - data source
 - configuring, 81
 - connecting via connection string, 86
 - connecting via logon dialog box, 85
 - data types, 89
 - INFORMIXDIR, 80
 - isolation levels, 92
 - ISQLT07C.DLL, 79, 80
 - locking levels, 92
 - ODBC conformance, 92
 - statements supported, 92
 - system requirements, 79
- Interactive Data Language (IDL), 10
- Intersolv
 - drivers, 12
- IS_CONNECTED keyword, 51
- IS_NULL keyword, 61
- IS_READONLY keyword, 52, 63
- isolation levels
 - INFORMIX, 92
 - Oracle, 107
 - Sybase, 125
- isolation levels and data consistency
 - compared, 154
 - dirty reads, 152
 - non-repeatable reads, 153

- phantom reads, [153](#)
- isolation levels, general, [152](#)
- isolation levels, specific
 - read committed, [153](#)
 - read uncommitted, [153](#)
 - repeatable read, [154](#)
 - serializable, [154](#)

J

joins. *See* syntax

K

keywords

- setting, [44](#)
- slash character use, [44](#)

L

language

- data manipulation language, [13](#)
- IDL, [10](#)
- Interactive Data Language, [10](#)
- SQL, [10](#)
- SQL syntax, [37](#)

LAST keyword, [66](#)

Level 1 API conformance, [12](#)

Level 2 API conformance, [12](#)

LIKE predicate, [38](#)

locking levels

- INFORMIX, [92](#)
- Oracle, [107](#)
- Sybase, [125](#)

locking modes and levels, [155](#)

M

MAX_CONNECTIONS keyword, [52](#)

MAX_RECORDSETS keyword, [52](#)

N

N_BUFFERS keyword, [56](#)

named variables, [43](#)

NEXT keyword, [66](#)

NULL_VALUE keyword, [61](#)

numeric functions, [164](#)

O

ODBC

- API functions, [12](#), [158](#)
- architecture, [10](#)
- conformance standards, [12](#)
- data source, [11](#)
- database application, [11](#)
- driver manager, [11](#)
- drivers, [11](#)
- error messages, [21](#)
- Open Database Connectivity, [10](#)
- outer join syntax, [38](#)
- scalar functions, [161](#)

ODBC Administrator

- using, [70](#)

ODBC conformance

- INFORMIX, [92](#)
- Oracle, [107](#)
- Sybase, [125](#)
- Text, [148](#)

ODBC Options, [72](#)

ODBC.INI file

- Data Source Specification section, [71](#)
- Default Data Source Specification section, [72](#)
- definition of, [69](#)
- file format, [71](#)
- modifying (UNIX), [70](#)
- ODBC Data Sources section, [71](#)

- ODBC Options section, [72](#)
- ODBC_LEVEL keyword, [52](#)
- Open Database Connectivity, [10](#)
- Oracle driver
 - connections supported, [108](#)
 - CORE35.DLL, [93](#)
 - CORE350.DLL, [93](#)
 - data source
 - connecting via connection string, [100](#)
 - configuring, [95](#)
 - connecting via logon dialog box, [100](#)
 - data types, [106](#)
 - isolation levels, [107](#)
 - locking levels, [107](#)
 - NLSRTL32.DLL, [93](#)
 - OCIW32.DLL, [93](#)
 - ODBC conformance, [107](#)
 - ORA73.DLL, [93](#)
 - ORACLE_HOME, [94, 95](#)
 - statements supported, [108](#)
 - system requirements, [93](#)

P

- parameters
 - formal, [43](#)
- PASSWORD keyword, [48](#)
- positional parameters, [43](#)
- PRIOR keyword, [66](#)
- procedure calling sequence, [42](#)
- procedure method calling sequence, [42](#)

R

- Recordset
 - moving within, [33](#)
 - using the cursor, [33](#)
- RECORDSET_SOURCE keyword, [64](#)
- RELATIVE keyword, [66](#)

S

- scalar functions, [37](#)
- scalar functions, ODBC, [161](#)
- Select statement
 - Text, [147](#)
- setting
 - keywords, [44](#)
 - named variables, [43](#)
- slash character , [44](#)
- SQL
 - core conformance level, [12](#)
 - core grammar, [13](#)
 - Data Sources dialog, [27](#)
 - extended conformance level, [12](#)
 - extended grammar, [13](#)
 - LIKE predicate, [38](#)
 - minimum conformance level, [12](#)
 - minimum grammar, [12](#)
 - syntax, [37](#)
 - using procedure calls instead, [38](#)
- SQL keyword, [56](#)
- sql.log file, [73](#)
- SQL_LEVEL keyword, [52](#)
- SQL_SERVER_NAME keyword, [52](#)
- statements supported
 - INFORMIX, [92](#)
 - Oracle, [108](#)
 - Sybase, [126](#)
 - Text, [148](#)
- string functions, [161](#)
- Structured Query Language, [10](#)
- Sybase driver
 - connections supported, [126](#)
 - data source
 - configuring, [110](#)
 - connecting via connection string, [119](#)
 - connecting via logon dialog box, [118](#)
 - data types, [124](#)
 - isolation levels, [125](#)
 - locking levels, [125](#)
 - ODBC conformance, [125](#)

- SQLEdit, 109
- statements supported, 126
- SYBASE, 109
- SYBPING, 109
- system requirements, 109
- syntax
 - arguments, 43
 - function methods, 43
 - functions, 42
 - outer join, 38
 - positional parameters, 43
 - procedure calls, 38
 - procedure methods, 42
 - procedures, 42
- system functions, 169
- system requirements
 - INFORMIX, 79
 - Oracle driver, 93
 - Sybase, 109
 - Text, 127

T

- TABLE keyword, 56
- table structure, defining
 - Text, 135
 - Text under UNIX, 138
- tables
 - connecting, 32
 - finding a specific table, 31
 - finding available tables, 31
 - GetTables method, 31
 - retrieving data, 32
 - working with data, 33
- terminology conventions, 16

- Text driver
 - Alter Table statement, 148
 - connections supported, 148
 - data source
 - configuring, 128
 - connecting via connection string, 142
 - data types, 147
 - date masks, 140
 - defining table structure, 135
 - defining table structure under UNIX, 138
 - formats, 127
 - ODBC conformance, 148
 - Select statement, 147
 - statements supported, 148
 - system requirements, 127
- time format, 37
- time functions, 166
- timestamp format, 37
- Trace file, 72

U

- UNIX
 - environment, 149
 - double-byte character sets, 151
 - system information file (.odbc.ini), 149
 - translators, 151
 - variables, 151
- USE_CURSOR_LIB keyword, 52
- USER_ID keyword, 48
- USER_NAME keyword, 52

V

- variable names, 43