IDL
DataMiner Guide

Legal and Copyright Notices

The IDL® and ENVI® software programs and the accompanying procedures, functions, and documentation described herein are sold under license agreements. Their use, duplication, and disclosure are subject to the restrictions stated in the license agreement. Harris Geospatial Solutions, Inc. reserves the right to make changes to this document at any time and without notice.

Limitation of Warranty

Harris Geospatial Solutions 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. Harris Geospatial Solutions shall not be liable for any direct, consequential, or any other damages, suffered by the licensed user or any other users resulting from the use of the software packages or the software documentation.

Permission to Reproduce Manuals

If you are a licensed user of Harris Geospatial Solutions software, Harris Geospatial Solutions grants you a limited, nontransferable license to reproduce its software's manuals 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 Harris Geospatial Solutions copyright notice.

Export Control Information

Harris Geospatial Solutions Software and its associated technology are subject to U.S. export controls including the United States Export Administration Regulations. The licensed user is responsible for ensuring compliance with all applicable U.S. export control laws and regulations. These laws include restrictions on destinations, end users and end use.

Copyright and Trademark Notices

IDL® and ENVI® are registered trademarks of Harris Corporation.

Esri®, ArcGIS®, ArcView®, and ArcInfo® are registered trademarks of Esri.

Adobe Illustrator® and Adobe PDF® Print Engine are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States and/or other countries.

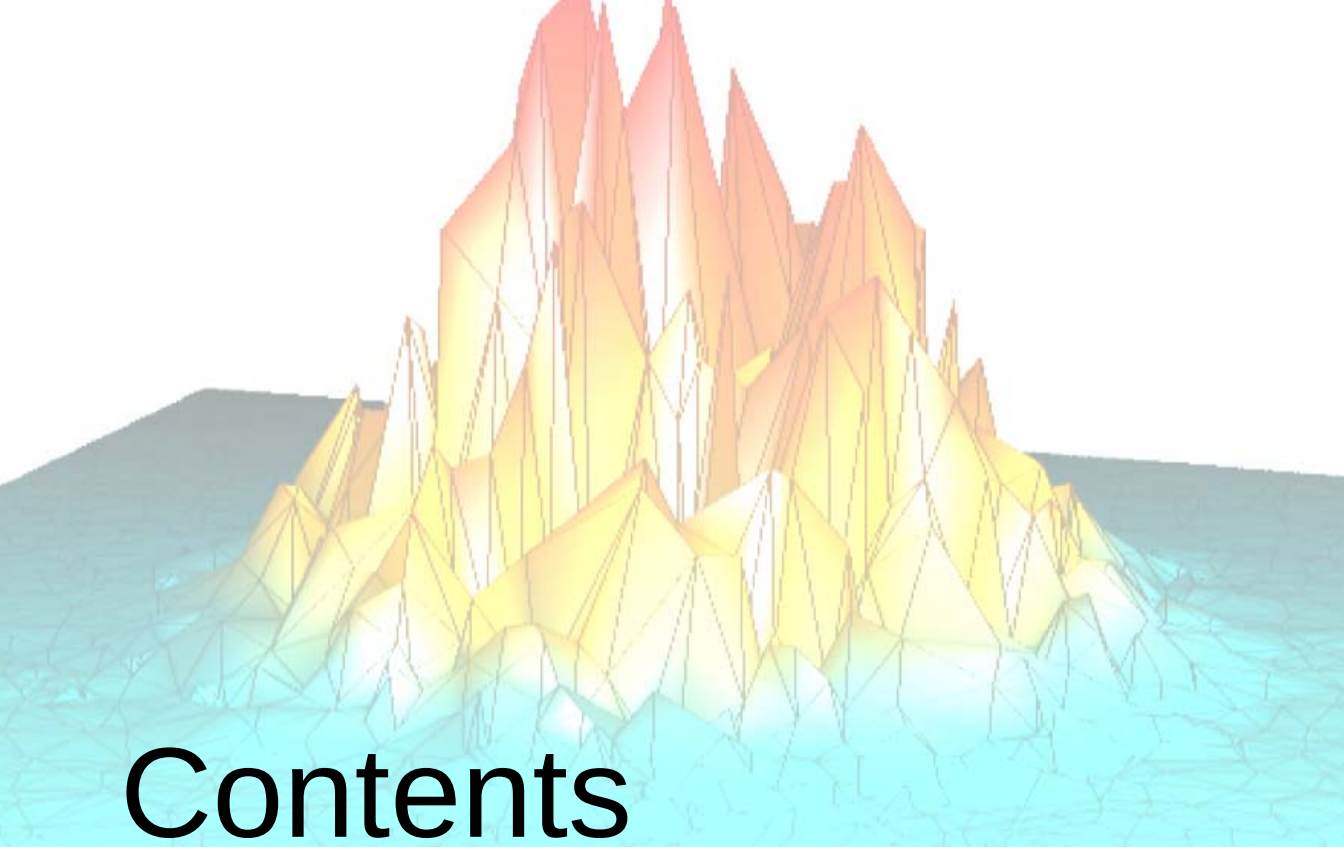
Macintosh® is a registered trademark of Apple Inc., registered in the U.S. and other countries.

PowerPoint®, PowerPoint icon and Windows® are registered trademarks of Microsoft Corporation in the United States and/or other countries.

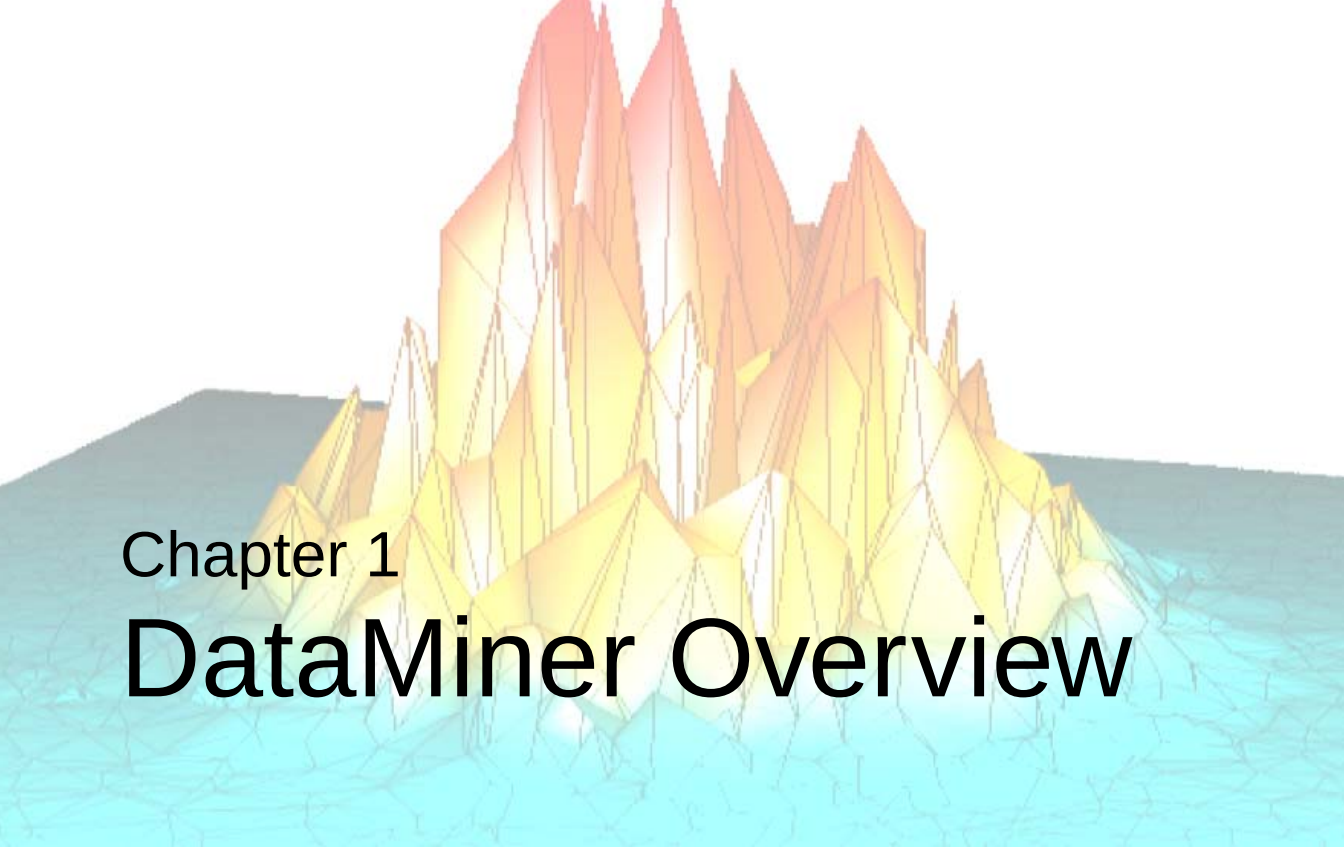
UNIX® is a registered trademark of The Open Group.

FLAASH® and QUAC® are registered trademarks of Spectral Sciences, Inc.

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



Contents



Chapter 1
DataMiner Overview

The following topics are discussed in this chapter:

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. We 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 stored in a database from within IDL. The IDL DataMiner allows you to do the following:

- Connect to a database management system (DBMS)
- Query 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 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.

About the DataMiner ODBC Drivers

The IDL DataMiner includes a set of ODBC drivers known as *Connect ODBC*, that is produced by DataDirect Technologies. Information about these drivers is available at:

`http://www.datadirect.com`

IDL version 7.1 supports version 5.3 [Service Pack 1] of the Connect ODBC drivers for most platforms.

The following tables describe the ODBC drivers that are included with and supported by the IDL DataMiner, by platform.

Note

DataDirect *Wire Protocol* drivers do not require that database client software be installed. All other drivers require that the appropriate client software be present. For more information on DataDirect drivers, specific platform requirements, issues, and how to configure the ODBC driver for use with your database, see the appropriate version of the *DataDirect Connect ODBC Reference* manual. See "About This Volume" on page 12 for further information on which manual is appropriate for your installation.

You must install version 2.6 or higher of the Microsoft Data Access Components (MDAC) on Windows platforms to use Connect ODBC drivers.

Driver	Database	Windows		Linux		Solaris Sparc [a]	
		32-bit	64-bit	32-bit	64-bit	32-bit	64-bit
Btrieve	Btrieve 6.5 Pervasive.SQL 7 Pervasive.SQL 2000	•					
DB2 Wire Protocol	7.1, 7.2, 8.1, 9.1, 9.5 DB2 Universal Database (UDB) 7.x, 8.1, 8.2, 9.1	•	•	•	•	•	•
dBASE	dBASE IV, V Clipper FoxPro 2.5, 2.6, 3.0 FoxPro 6.0 with 3.0 functionality only FoxPro 3.0 database container (DBC)	•		•		•	
Informix (client)	SE 7.2 Dynamic Server 9.2, 9.3, 9.4x, 10, 11	•				•	
Informix Wire Protocol	Dynamic Server 9.2, 9.3, 9.4, 10	•	•	•	•	•	

Table 1-1: Database Support by Platform, DataDirect version 5.3

Driver	Database	Windows		Linux		Solaris Sparc [a]	
		32-bit	64-bit	32-bit	64-bit	32-bit	64-bit
MySQL Wire Protocol	MySQL 5.0.x server	•	•	•	•	•	•
Oracle (client)	7.3.4+ (w/Net8 Client) 8.0.5+ 8i R1, R2, R3 9i R1, R2 10g R1, 11g	•	•	•	•	• [b]	
Oracle Wire Protocol	8i R2, R3 9i R1, R2 10g R1, R2 11g	•	•	•	•	•	•
Paradox	4, 5, 7, 8, 9, 10	•					
SQL Server Wire Protocol	7.0 2000 (with Service Packs 1, 2, 3, 3a, 4) 2000 Desktop Engine 2000 Enterprise Edition (64-bit) 2005	•	•	•	•	•	•
Sybase Wire Protocol	Adaptive Server 11.5+ Adaptive Server Enterprise 12.0, 12.5x, 15	•	•	•	•	•	•
ASCII Text	n/a	•		•		•	
XML	n/a	•					

Table 1-1: Database Support by Platform, DataDirect version 5.3 (Continued)

[a] DataMiner is not available on Solaris X86 platforms

[b] SUSE Enterprise Server 10.0 or 9.0 only

About This Volume

The *IDL DataMiner Guide* describes IDL's ODBC interface. Information about installation and configuration of the specific ODBC drivers provided with the DataMiner system is supplied by the driver manufacturer. The manufacturer's documentation is provided in the `help/DataDirect` subdirectory of the IDL installation directory.

To view the the *DataDirect Connect for ODBC* Help for 32-bit systems, use a web browser to open the file

```
<IDL_DIR>/help/DataDirect/help.htm
```

For 64-bit systems, open

```
<IDL_DIR>/help/DataDirect64/help.htm
```

where *<IDL_DIR>* is the path to your IDL installation.

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, and conventions. This chapter also provides information about ODBC conformance levels and network access requirements, and discusses UNIX-specific installation issues.
- 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.
- Appendix A, "ODBC Configuration on UNIX Systems", explains the ODBC initialization file and its contents.

Terminology

The following terms 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.

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.

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 Appendix C, “ODBC API and Scalar Functions” in the *DataDirect Connect ODBC Reference* 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 Connect ODBC 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 for the database you are connecting to in the *Connect ODBC Reference*.

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 Connect ODBC 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

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

IDL DataMiner system components, including ODBC drivers, are installed automatically by the IDL installation program (assuming you elected to install the DataMiner components when installing IDL). On UNIX systems, you may need to make some of the following additional modifications; at the very least, you will need to copy the default ODBC initialization file created by the installer to your $HOME directory.

ODBC Initialization File

During the installation process, the IDL installation program creates a template ODBC initialization file. This file describes the installed ODBC drivers and allows you to configure the drivers to suit your needs.

The template file contains information about the ODBC drivers installed on your system by IDL, but nothing about your specific data sources. At the very least, you will need to add the data source names to the ODBC initialization file; depending on your specific installation, you may also need to add details such as the database host name and database instance name. You might also wish to make database access more convenient by specifying login or password information. You will also need to either copy the template ODBC initialization file to your $HOME directory or create an environment variable pointing to its location. See Appendix A, “ODBC Configuration on UNIX Systems” for details.

Oracle ODBC Drivers

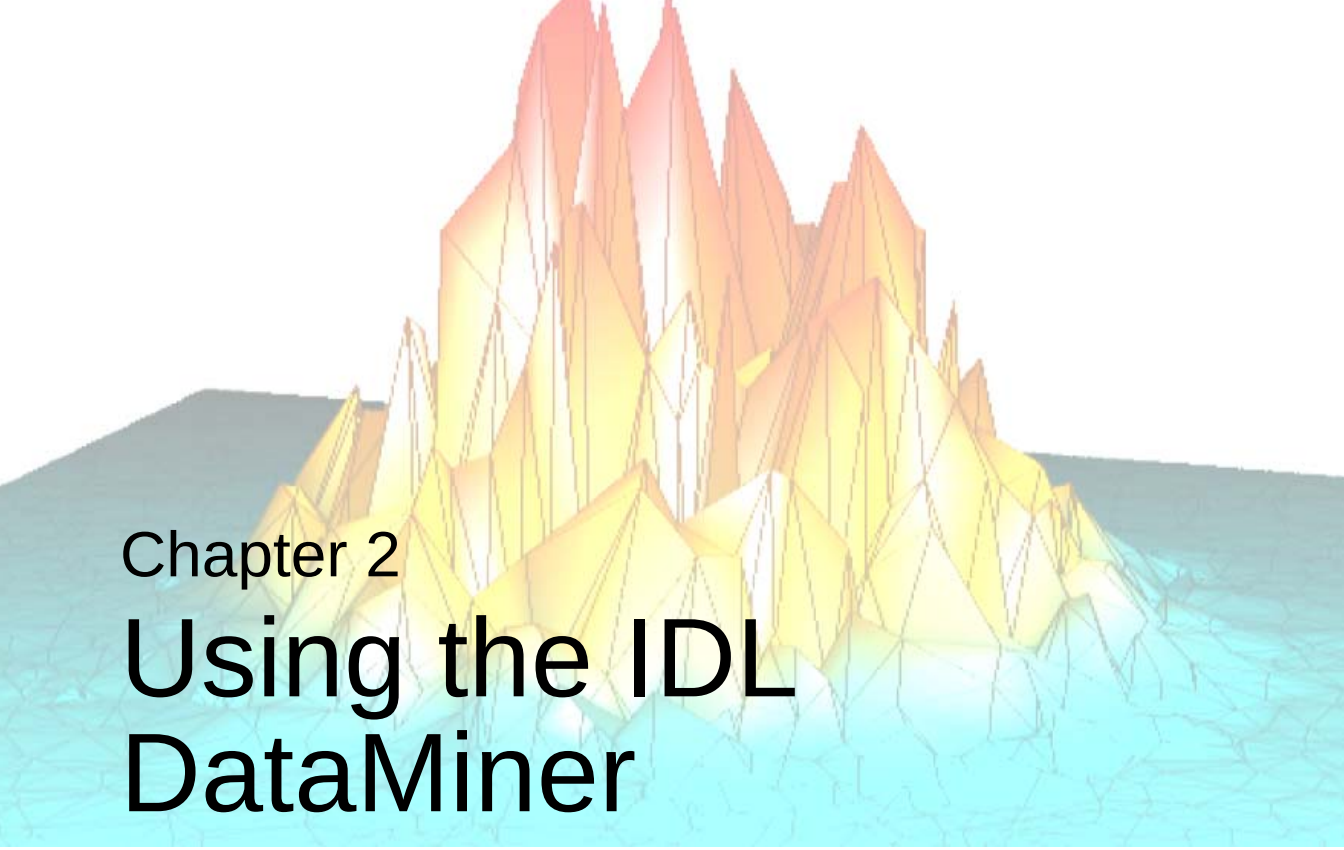
On some UNIX systems, the Oracle ODBC drivers must be linked against portions of the Oracle installation. For more information on how this is performed consult the files located in the Dataminer directory:

```
$IDL_DIR/bin/bin.platform/dm/src/oracle
```

where *platform* is the name of your operating system.

Note

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



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

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 *Application Programming* 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 58 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:

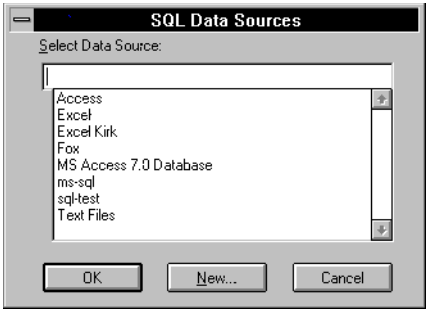


Figure 2-1: SQL Data Sources (Windows dialog)

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.

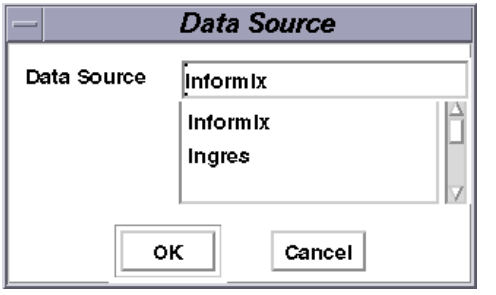


Figure 2-2: SQL Data Sources (Motif dialog)

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.

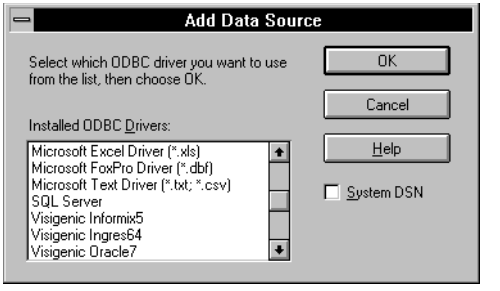


Figure 2-3: Add Data Source

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 22, 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 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 60 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, the 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, tables.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.

Reserved ODBC SQL Words

While using the IDLdbDatabase::ExecuteSQL method in Dataminer, do not use the following reserved words in the SQL command string: DOUBLE, FLOAT, and TEMP. These words are reserved in ODBC SQL and result in syntax errors if you attempt to use them in your SQL code.

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.

Data Type 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 2-2 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 2-2: IDL - SQL Type Code Mapping

Error Messages

The error messages returned by the IDL DataMiner 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 63 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.

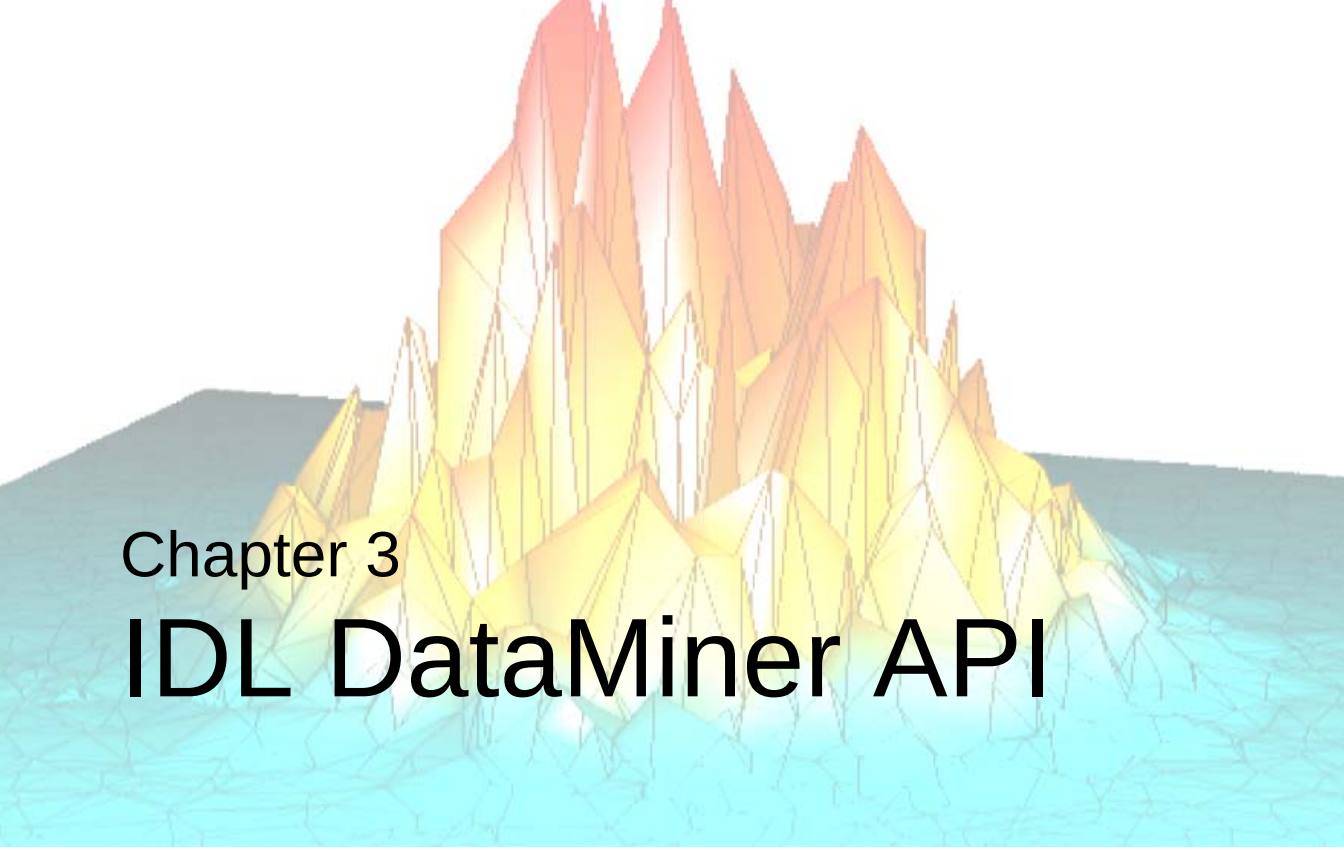
<error code> is the error code returned from the data source. This code is a native error of the datasource and describes the error condition that the datasource detected.

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 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.

Note

IDL DataMiner must be licensed on your system to be able to use these functions and objects.

Syntax

The “Syntax” 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 procedure (see “OBJ_DESTROY” (*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.

Note

Due to Motif library inconsistencies, this dialog may fail on SUN Solaris systems.

Syntax

status = DIALOG_DBCONNECT(*DBobj* [, DATASOURCE=*string*]
[, USER_ID=*string*] [, PASSWORD=*string*]
[, DIALOG_PARENT=*widget id*])

Return Value

The function returns true (1) unless you selected the dialog's **Cancel** button, in which case it returns false (0).

Arguments

DBobj

A valid IDLdbDatabase 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.

Version History

5.0	Introduced

DB_EXISTS

Use the DB_EXISTS function to determine if the database functionality is available on a specific platform.

Syntax

status = DB_EXISTS()

Return Value

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.

Arguments

None.

Keywords

None.

Version History

5.0	Introduced

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).

Superclasses

None

Creation

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.

Properties

Objects of this class have the following properties. See "IDLdbDatabase Properties" on page 48 for details on individual properties.

- CAN_GET_TABLES
- DBMS_NAME
- DRIVER_ODBC_LEVEL
- DBMS_VERSION
- DRIVER_VERSION
- IS_CONNECTED
- IS_READONLY
- MAX_CONNECTIONS
- MAX_RECORDSETS
- ODBC_LEVEL
- SQL_LEVEL
- SQL_SERVER_NAME

- USE_CURSOR_LIB
- USER_NAME
- VERBOSE

Methods

This class has the following methods:

- "IDLdbDatabase::Cleanup" on page 53
- "IDLdbDatabase::Connect" on page 54
- "IDLdbDatabase::ExecuteSQL" on page 56
- "IDLdbDatabase::GetDatasources" on page 58
- "IDLdbDatabase::GetProperty" on page 59
- "IDLdbDatabase::GetTables" on page 60
- "IDLdbDatabase::Init" on page 61
- "IDLdbDatabase::SetProperty" on page 63

Version History

5.0	Introduced

IDLdbDatabase Properties

IDLdbDatabase objects have the following properties in addition to properties inherited from any superclasses. Properties with the word “Yes” in the “Get” column of the property table can be retrieved via IDLdbDatabase::GetProperty. Properties with the word “Yes” in the “Set” column in the property table can be set via IDLdbDatabase::SetProperty.

Note

For a discussion of the property description tables shown below, see “About Object Property Descriptions” (Chapter 28, *IDL Reference Guide*).

CAN_GET_TABLES

If True, the GetTables method is available for the current driver.

Property Type	Boolean		
Name String	*not displayed*		
Get: Yes	**Set:** No	**Init:** No	**Registered:** No

DBMS_NAME

The name of the Database with which the object is associated.

Property Type	String		
Name String	*not displayed*		
Get: Yes	**Set:** No	**Init:** No	**Registered:** No

DRIVER_ODBC_LEVEL

The ODBC level supported by the driver being used to connect to the database.

Property Type	String		
Name String	*not displayed*		
Get: Yes	**Set:** No	**Init:** No	**Registered:** No

DBMS_VERSION

The version number of the Database that the object is associated with.

Property Type	String		
Name String	*not displayed*		
Get: Yes	**Set:** No	**Init:** No	**Registered:** No

DRIVER_VERSION

The version number of the ODBC driver being used to connect to the database.

Property Type	String		
Name String	*not displayed*		
Get: Yes	**Set:** No	**Init:** No	**Registered:** No

IS_CONNECTED

If True, a connection to a database exists.

Property Type	Boolean		
Name String	*not displayed*		
Get: Yes	**Set:** No	**Init:** No	**Registered:** No

IS_READONLY

If True, the database is read-only.

Property Type	Boolean		
Name String	*not displayed*		
Get: Yes	**Set:** No	**Init:** No	**Registered:** No

MAX_CONNECTIONS

The maximum number of connections supported by the ODBC driver. If the maximum value is unknown, this property will contain 0.

Property Type	Integer		
Name String	*not displayed*		
Get: Yes	**Set:** No	**Init:** No	**Registered:** No

MAX_RECORDSETS

The maximum number of recordsets supported by the ODBC driver. If the maximum is unknown, this property will contain 0.

Property Type	Integer		
Name String	*not displayed*		
Get: Yes	**Set:** No	**Init:** No	**Registered:** No

ODBC_LEVEL

The ODBC level of the driver manager.

Property Type	String		
Name String	*not displayed*		
Get: Yes	**Set:** No	**Init:** No	**Registered:** No

SQL_LEVEL

The SQL level supported by the connection.

Property Type	String		
Name String	*not displayed*		
Get: Yes	**Set:** No	**Init:** No	**Registered:** No

SQL_SERVER_NAME

The SQL server name for this database connection.

Property Type	String		
Name String	*not displayed*		
Get: Yes	**Set:** No	**Init:** No	**Registered:** No

USE_CURSOR_LIB

If True, the ODBC cursor library will be used. 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, so this property may contain a True value if the USE_CURSOR_LIB property was set, *even if the cursor library was not subsequently loaded*.

Property Type	BOOLEAN		
Name String	*not displayed*		
Get: Yes	**Set:** Yes	**Init:** No	**Registered:** No

USER_NAME

The user name used during the connection to the datasource.

Property Type	String		
Name String	*not displayed*		
Get: Yes	**Set:** No	**Init:** No	**Registered:** No

VERBOSE

If True, verbose error messages will be generated. 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.

Property Type	BOOLEAN		
Name String	*not displayed*		
Get: No	**Set:** Yes	**Init:** No	**Registered:** No

IDLdbDatabase::Cleanup

The IDLdbDatabase::Cleanup procedure method performs all cleanup on the object.

Note

Cleanup methods are special *lifecycle methods*, and as such cannot be called outside the context of object creation and destruction. This means that in most cases, you cannot call the Cleanup method directly. There is one exception to this rule: if you write your own subclass of this class, you can call the Cleanup method from within the Init or Cleanup method of the subclass.

Syntax

OBJ_DESTROY, *Obj*

or

Obj->[IDLdbDatabase::]Cleanup *(In a lifecycle method only.)*

Arguments

None

Keywords

None

Version History

5.0	Introduced

IDLdbDatabase::Connect

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

Syntax

DBobj -> [IDLdbDatabase::]Connect [, CONNECTION=*string*]
[, DATASOURCE=*string*] [, USER_ID=*string*] [, PASSWORD=*string*]

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. Depending on the database you use, you may be able to specify a default data source. See the *Connect ODBC Reference* section for your database or Appendix A, “ODBC Configuration on UNIX Systems”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.

Version History

5.0	Introduced

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 32for information on some common questions about ODBC SQL syntax.

Syntax

DBobj -> [IDLdbDatabase::]ExecuteSQL, *strSQL*

Arguments

Note

Do not use the following words in the argument string: DOUBLE, FLOAT, and TEMP. These words are reserved in SQL.

strSQL

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

If *strSQL* is a command that produces a return value, that value will be ignored. If you want to capture the return of an SQL statement (or Stored Procedure) that provides a return value, see “SQL” on page 65.

Note

Always enclose the string value in double quotes.

Keywords

None.

Version History

5.0	Introduced

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.

Syntax

Datasources = *DBobj* -> [IDLdbDatabase::]GetDatasources()

Return Value

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.

Arguments

None.

Keywords

None.

Version History

5.0	Introduced

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.

Syntax

DBobj -> [IDLdbDatabase::]GetProperty [, *PROPERTY=variable*]

Arguments

None.

Keywords

Any property listed under “IDLdbDatabase Properties” on page 48 that contains the word “Yes” in the “Get” column of the properties table can be retrieved using this method. To retrieve the value of a property, specify the property name as a keyword set equal to a named variable that will contain the value of the property.

Version History

5.0	Introduced

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.

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.

Syntax

Tables = *DBobj* -> [IDLdbDatabase::]GetTables()

Return Value

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.

Arguments

None.

Keywords

None.

Version History

5.0	Introduced

IDLdbDatabase::Init

The IDLdbDatabase::Init function method initializes a database object.

Note

Init methods are special *lifecycle methods*, and as such cannot be called outside the context of object creation. This means that in most cases, you cannot call the Init method directly. There is one exception to this rule: if you write your own subclass of this class, you can call the Init method from within the Init method of the subclass.

Syntax

Obj = OBJ_NEW('IDLdbDatabase' [, *PROPERTY*=*value*])

or

Result = *Obj*->[IDLdbDatabase::]Init([, *PROPERTY*=*value*]) *(In a lifecycle method only.)*

Return Value

When this method is called indirectly, as part of the call to the OBJ_NEW function, the return value is an object reference to the newly-created object.

When called directly within a subclass Init method, the return value is 1 if initialization was successful, or zero otherwise.

Arguments

None.

Keywords

Any property listed under “IDLdbDatabase Properties” on page 48 that contains the word “Yes” in the “Init” column of the properties table can be initialized during object creation using this method. To initialize the value of a property, specify the property name as a keyword set equal to the appropriate property value.

Version History

5.0	Introduced

IDLdbDatabase::SetProperty

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

Note

You must connect to the data source associated with the database object (using the IDLdbDatabase::Connect method) before attempting to set properties on the database object.

Syntax

DBobj -> [IDLdbDatabase::]SetProperty [, *PROPERTY*=*value*]

Arguments

None.

Keywords

Any property listed under “IDLdbDatabase Properties” on page 48 that contains the word “Yes” in the “Set” column of the properties table can be set using this method. To set the value of a property, specify the property name as a keyword set equal to the appropriate property value.

Version History

5.0	Introduced

IDLdbRecordset

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

Superclasses

None

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*, [, N_BUFFERS=*integer*] [, SQL=*string* | TABLE=*string*])

where *DBobj* is the object reference of the database object and either the SQL or the TABLE property must be specified.

N_BUFFERS

Set this keyword equal to the number of records to store per database read request in a rapid access memory location. When a IDLdbRecordset object requests records from the datasource it is associated with, the desired records are retrieved from the datasource and cached in the recordset object. The operation to request and transfer the desired records from the datasource can be fairly time consuming, which can impact performance of record access when a large number of records are being requested. Setting this to the optimal number of records can greatly increase IDLdbRecordset::MoveCursor performance and, thus, the overall speed of IDL DataMiner. A higher value is particularly useful, if you are looping through large IDLdbRecordset's with MoveCursor(/NEXT) or MoveCursor(/PREVIOUS) calls. A relatively low value might be called for, if your incremental Dataminer calls are searching for records that are widely dispersed in the target table. The default value is 10 records.

Transferring records from the datasource to the recordset object in a block can drastically increase performance, especially when accessing records in a sequential order. The N_BUFFERS keyword gives the user the ability to modify the size of the block of records the dataminer will transfer from the data sources when a request is being made by the recordset. Increasing the block/buffer size can dramatically increase record access time. You may need to experiment with different 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. This SQL statement can be a Stored Procedure call that provides a return value.

Note

If the SQL keyword uses SQL statement syntax unrecognized by the DBMS or the ODBC driver, IDL throws an error that stops execution of the program if not caught and handled by the Dataminer programmer.

Note

If the SQL keyword uses an SQL statement that has acceptable syntax, but returns NULL data (because it has filtered out everything in the table it is referencing), then NO ERROR is thrown. The object returned is a valid IDL object. To determine the object returned was actually an “empty” recordset, you must test the recordset with a subsequent call of:

```
result = RSObj->MoveCursor(/FIRST)
```

if `result eq 0` (or `result ne 1`), then the recordset returned by OBJ_NEW was an empty (or NULL) recordset.

TABLE

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

Note

If the TABLE keyword references a table that does not exist, then IDL throws an error that stops execution of the program if not caught and handled by the Dataminer programmer.

Properties

Objects of this class have the following properties. See “IDLdbRecordset Properties” on page 67 for details on individual properties.

- CAN_MOVE_ABSOLUTE
- CAN_MOVE_FIRST
- CAN_MOVE_LAST

- CAN_MOVE_NEXT
- CAN_MOVE_PRIOR
- CAN_MOVE_RELATIVE
- FIELD_INFO
- GET_DATABASE
- IS_READONLY
- N_BUFFERS
- RECORDSET_SOURCE

Methods

- "IDLdbRecordset::AddRecord" on page 71
- "IDLdbRecordset::Cleanup" on page 72
- "IDLdbRecordset::CurrentRecord" on page 73
- "IDLdbRecordset::DeleteRecord" on page 74
- "IDLdbRecordset::GetField" on page 75
- "IDLdbRecordset::GetProperty" on page 76
- "IDLdbRecordset::GetRecord" on page 77
- "IDLdbRecordset::Init" on page 79
- "IDLdbRecordset::MoveCursor" on page 81
- "IDLdbRecordset::NFields" on page 83
- "IDLdbRecordset::SetField" on page 84

Version History

5.0	Introduced

IDLdbRecordset Properties

IDLdbRecordset objects have the following properties in addition to properties inherited from any superclasses. Properties with the word “Yes” in the “Get” column of the property table can be retrieved via IDLdbRecordset::GetProperty. There are no Init or SetProperty methods for the IDLdbRecordset object.

Note

For a discussion of the property description tables shown below, see “About Object Property Descriptions” (Chapter 28, *IDL Reference Guide*).

CAN_MOVE_ABSOLUTE

If True, the cursor for the recordset can move to an absolute record number.

Property Type	Boolean		
Name String	*not displayed*		
Get: Yes	**Set:** No	**Init:** No	**Registered:** No

CAN_MOVE_FIRST

If True, the cursor for the recordset can move to the first record.

Property Type	Boolean		
Name String	*not displayed*		
Get: Yes	**Set:** No	**Init:** No	**Registered:** No

CAN_MOVE_LAST

If True, the cursor for the recordset can move to the last record.

Property Type	Boolean		
Name String	*not displayed*		
Get: Yes	**Set:** No	**Init:** No	**Registered:** No

CAN_MOVE_NEXT

If True, the cursor for the recordset can move to the next record.

Property Type	Boolean		
Name String	*not displayed*		
Get: Yes	**Set:** No	**Init:** No	**Registered:** No

CAN_MOVE_PRIOR

If True, the cursor for the recordset can move to the previous record.

Property Type	Boolean		
Name String	*not displayed*		
Get: Yes	**Set:** No	**Init:** No	**Registered:** No

CAN_MOVE_RELATIVE

If True, the cursor for the recordset can move to a record number relative to the current record number.

Property Type	Boolean		
Name String	*not displayed*		
Get: Yes	**Set:** No	**Init:** No	**Registered:** No

FIELD_INFO

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 property 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.

Property Type	Array of structures		
Name String	*not displayed*		
Get: Yes	**Set:** No	**Init:** No	**Registered:** No

GET_DATABASE

An object reference to the database object used when the current recordset object was created.

Property Type	Object reference		
Name String	*not displayed*		
Get: Yes	**Set:** No	**Init:** No	**Registered:** No

IS_READONLY

If True, the table is read-only.

Property Type	Boolean		
Name String	*not displayed*		
Get: Yes	**Set:** No	**Init:** No	**Registered:** No

N_BUFFERS

The number of buffers allocated for the recordset.

Property Type	Integer		
Name String	*not displayed*		
Get: Yes	**Set:** No	**Init:** Yes	**Registered:** No

RECORDSET_SOURCE

A string containing either the table name or SQL statement used to create the recordset.

Property Type	String		
Name String	*not displayed*		
Get: Yes	**Set:** No	**Init:** No	**Registered:** No

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.

Syntax

RSobj -> [IDLdbRecordset::]AddRecord*[, field$_1$[, field$_2$[, ...[, field$_n$]]]]*
[, SET_AUTOINCREMENT=*integer*]

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.

Note

When using the cursor library, adding a new record to a table that only contains a single autoincrement field can result in an error. To add a record, either set the SET_AUTOINCREMENT keyword, or do not use an autoincrement field.

Version History

5.0	Introduced

IDLdbRecordset::Cleanup

The IDLdbRecordset::Cleanup procedure method performs all cleanup on the object.

Note

Cleanup methods are special *lifecycle methods*, and as such cannot be called outside the context of object creation and destruction. This means that in most cases, you cannot call the Cleanup method directly. There is one exception to this rule: if you write your own subclass of this class, you can call the Cleanup method from within the Init or Cleanup method of the subclass.

Syntax

OBJ_DESTROY, *Obj*

or

Obj->[IDLdbRecordset::]Cleanup *(In a lifecycle method only.)*

Arguments

None

Keywords

None

Version History

5.0	Introduced

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.

Syntax

number = *RSobj* -> [IDLdbRecordset::]CurrentRecord()

Arguments

None.

Keywords

None.

Version History

5.0	Introduced

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.

Syntax

RSobj -> [IDLdbRecordset::]DeleteRecord

Arguments

None.

Keywords

None.

Version History

5.0	Introduced

IDLdbRecordset::GetField

Use the IDLdbRecordset::GetField function method to get the value of a field from the current record in the recordset.

Syntax

value = *RSobj* -> [IDLdbRecordset::]GetField(*iFieldNumber* [, IS_NULL=*variable*] [, NULL_VALUE=*variable*])

Return Value

Returns the specified field value or if the value of the field is NULL (as defined by SQL) a null value (zero or an empty string) is returned.

Arguments

iFieldNumber

The number of the field for which the value will be returned. Field numbers have a range of 0 <= *n* < 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.

Version History

5.0	Introduced

IDLdbRecordset::GetProperty

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

Syntax

RSobj -> [IDLdbRecordset::]GetProperty [, *PROPERTY=variable*]

Arguments

None.

Keywords

Any property listed under “IDLdbRecordset Properties” on page 67 that contains the word “Yes” in the “Get” column of the properties table can be retrieved using this method. To retrieve the value of a property, specify the property name as a keyword set equal to a named variable that will contain the value of the property.

Version History

5.0	Introduced

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 or HEAP_FREE routine.

Syntax

Result = *RSobj* -> [IDLdbRecordset::]GetRecord()

Return Value

Returns the value of the current record in an IDL anonymous structure.

Arguments

None.

Keywords

None.

Examples

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
```

Version History

5.0	Introduced

IDLdbRecordset::Init

The IDLdbRecordset::Init function method initializes a database object.

Note

Init methods are special *lifecycle methods*, and as such cannot be called outside the context of object creation. This means that in most cases, you cannot call the Init method directly. There is one exception to this rule: if you write your own subclass of this class, you can call the Init method from within the Init method of the subclass.

Syntax

Obj = OBJ_NEW('IDLdbRecordset' [, *PROPERTY=value*])

or

Result = *Obj*->[IDLdbRecordset::]Init([, *PROPERTY=value*]) *(In a lifecycle method only.)*

Return Value

When this method is called indirectly, as part of the call to the OBJ_NEW function, the return value is an object reference to the newly-created object.

When called directly within a subclass Init method, the return value is 1 if initialization was successful, or zero otherwise.

Arguments

None.

Keywords

Any property listed under “IDLdbRecordset Properties” on page 67 that contains the word “Yes” in the “Init” column of the properties table can be initialized during object creation using this method. To initialize the value of a property, specify the property name as a keyword set equal to the appropriate property value.

Version History

5.0	Introduced

IDLdbRecordset::MoveCursor

Use the IDLdbRecordset::MoveCursor function method to move the cursor in a given recordset.

Syntax

Result = *RSobj* -> [IDLdbRecordset::]MoveCursor([, ABSOLUTE=*integer*] [, /FIRST] [, /LAST] [, /NEXT] [, /PRIOR] [, RELATIVE=*integer*])

Return Value

The function returns true (1) if the move operation was successful, or false (0) otherwise.

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.

Version History

5.0	Introduced

IDLdbRecordset::NFields

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

Syntax

status = *RSobj* -> [IDLdbRecordset::]NFields()

Return Value

Returns the number of fields.

Arguments

None.

Keywords

None.

Version History

5.0	Introduced

IDLdbRecordset::SetField

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

Syntax

RSobj -> [IDLdbRecordset::]SetField, *iFieldNumber*, *Value* [, /NULL]

Arguments

iFieldNumber

The number of the field whose value is returned. Field numbers have a range of 0 <= *n* < 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.

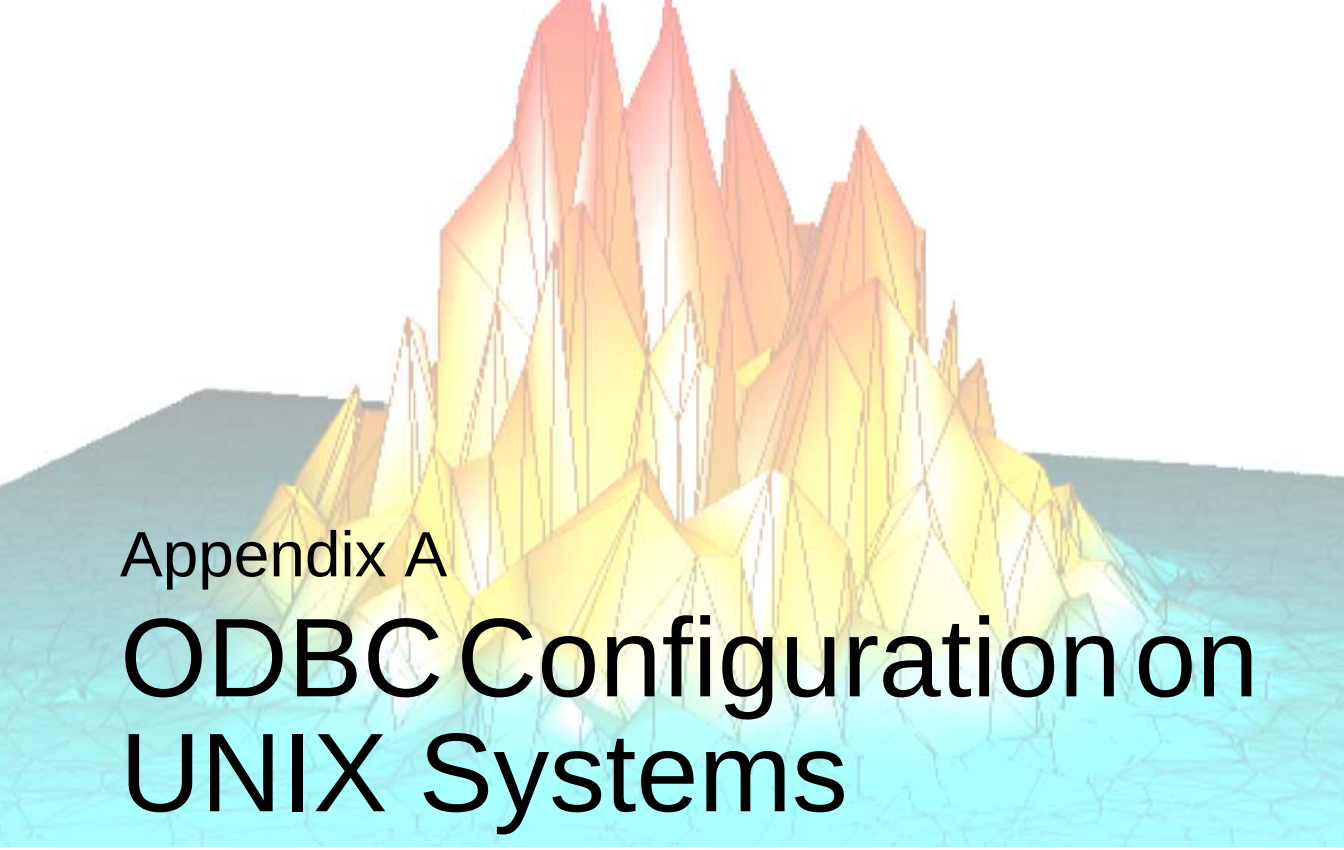
Keywords

NULL

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

Version History

5.0	Introduced



Appendix A

ODBC Configuration on UNIX Systems

The following topics are discussed in this chapter:

Overview of ODBC Configuration

The ODBC initialization file is used by the ODBC Driver Manager and ODBC drivers. Although ODBC initialization information exists on both Windows and UNIX systems, on Windows systems the information resides in the Registry, and all changes are made using the Windows ODBC Administrator (launched by clicking on the ODBC Data Sources icon in the Windows Control Panel) as described in the *Connect ODBC Reference*.

On UNIX systems, the ODBC initialization file is a text file that must be modified manually to reflect the data sources available in your environment. During the IDL installation process, a template initialization file named `odbc.ini` is installed in the *`IDL_DIR`*`/resource/dm/`*`os`*`/` directory, where *`IDL_DIR`* is the root directory of the IDL distribution and *`os`* is the directory named after your particular operating system. This file contains information about the ODBC drivers installed along with the DataMiner.

You must modify the ODBC initialization file reflect your system's data source configuration, and you must make the file available to all IDL DataMiner users on the system. Do the following:

1. Create a backup copy of the template *`IDL_DIR`*`/resource/dm/`*`os`*`/odbc.ini` file named `odbc.ini.orig`.
2. Modify the template `odbc.ini` file to reflect your system's data source configuration. The format of the initialization file is described in "ODBC Initialization File Format" on page 88. See the *Connect ODBC Reference* for information on the specific ODBC drivers used at your site.
3. Make the initialization file available to IDL DataMiner users. You can do this in either of the following ways:
   - Create an environment variable named `ODBCINI` that contains the path to the initialization file in each IDL DataMiner user's environment. This method is useful if every IDL DataMiner user uses the same datasource configuration.
   - Copy the modified `odbc.ini` file to each IDL DataMiner user's home directory, using `.odbc.ini` as the filename (note the initial dot). This method is useful if different IDL DataMiner users use different datasource configurations.

Note

When the IDL DataMiner starts, it looks first for the `ODBCINI` environment variable. If the variable exists, the DataMiner will use the file it refers to and ignore any `.odbc.ini` file in the user's `$HOME` directory.

About the Default ODBC Configuration

The default *`IDL_DIR`*`/resource/dm/`*`os`*`/odbc.ini` file contains information about the ODBC drivers installed on your system by IDL, but nothing about your specific data sources. At the very least, you will need to add the data source names to the ODBC initialization file; depending on your specific installation, you may also need to add details such as the database host name, database instance name. You might also wish to make database access more convenient by specifying login or password information.

Changing the ODBC Configuration

If you change the values in the ODBC initialization file while the IDL DataMiner is in use, you must destroy any existing database object and reconnect before the changes will be apparent. See "Destroying Database Objects" on page 42 for details on destroying a database object.

ODBC Initialization File Format

The ODBC initialization 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 initialization file 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 initialization file.

The `path_specification` is the full path to the driver shared library.

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=IDL_DIR/bin/bin.platform/dm/drivers/dmsyb13.so
Server=SYBASE10
Database=agencies
UID=marvin
```

where *`IDL_DIR`* is the root directory of the IDL distribution and *`platform`* is the platform-specific `bin` directory, for example `solaris2.sparc`.

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=IDL_DIR/bin/bin.platform/dm/drivers/dmor713.so
Server=t:mickey:customers
UID=marvin
```

where *`IDL_DIR`* is the root directory of the IDL distribution and *`platform`* is the platform-specific `bin` directory, for example `solaris2.sparc`.

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 (`[ODBC]`) specifies the ODBC root directory 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

This section of the ODBC initialization file is recommended 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.

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=IDL_DIR/bin/bin.platform/dm
Trace=1
TraceFile=IDL_DIR/bin/bin.platform/dm/drivers/trace.log
TraceDll=IDL_DIR/bin/bin.platform/dm/lib/odbctrac.so
```

where *IDL_DIR* is the root directory of the IDL distribution and *platform* is the platform-specific `bin` directory, for example `solaris2.sparc`.

ODBC Initialization File Example

The following example shows a UNIX ODBC initialization file.

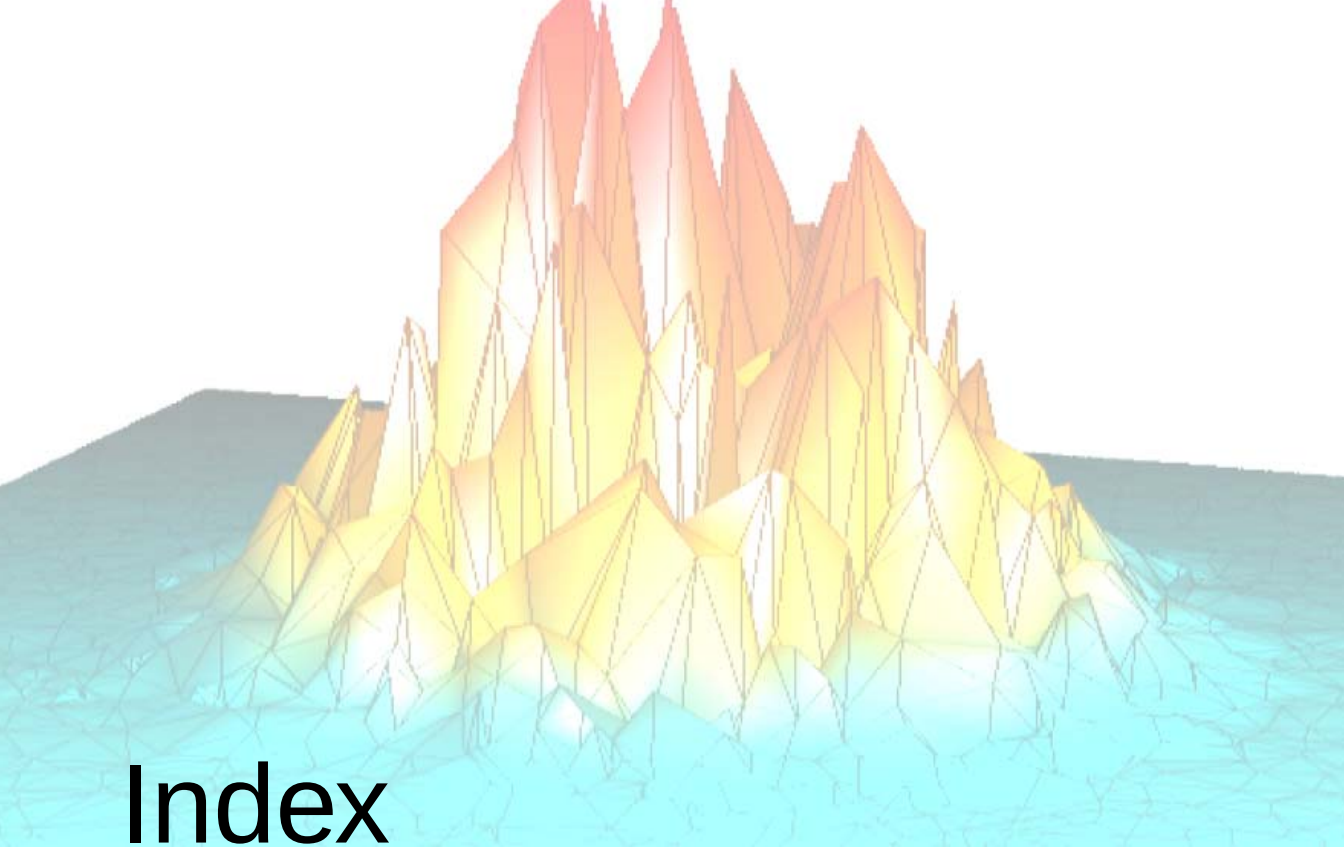
```
[ODBC Data Sources]
Informix9=INTERSOLV 3.11 Informix 9 Driver
Text=INTERSOLV 3.11 Text Driver

[Text]
Driver=IDL_DIR/bin/bin.platform/dm/lib/dmtxt13.so
Description=Text driver
Database=/home/kirk/dmtest
AllowUpdateAndDelete=1

[Informix9]
Driver=IDL_DIR/bin/bin.platform/dm/lib/dminf913.so
Description=Informix9
Database=odbc
HostName=informixhost
LogonID=odbc01
Password=odbc01

[ODBC]
InstallDir=IDL_DIR/bin/bin.platform/dm
Trace=1
TraceFile=IDL_DIR/bin/bin.platform/dm/drivers/trace.log
TraceDll=IDL_DIR/bin/bin.platform/dm/lib/odbctrac.so
```

where *IDL_DIR* is the root directory of the IDL distribution and *platform* is the platform-specific `bin` directory, for example `solaris2.sparc`.



Index

D

E

F

I

T