

SQLBase

Guide to Connecting to SQLBase

20-6245-0001



Trademarks

Gupta, the Gupta logo, Centura, the Centura logo, Centura net.db, Centura Web Developer, Gupta Powered, the Gupta Powered logo, Fast Facts, Object Nationalizer, Quest, QuickObjects, SQL/API, SQLBase, SQLBase Exchange, SQLConsole, SQLGateway, SQLHost, SQLNetwork, SQLRouter, SQLTalk and Team Object Manager are trademarks of Gupta Technologies LLC and may be registered in the United States of America and/or other countries. The trademarks TeamWindows, ReportWindows and EditWindows, and the registered trademark SQL Windows, are all exclusively used and licensed by Gupta Technologies LLC.

Adobe is a trademark of Adobe Systems, Incorporated.

IBM, OS/2, NetBIOS, and AIX are registered trademarks of International Business Machines Corporation.

Java and Solaris are trademarks of Sun Microsystems, Incorporated.

Microsoft, Internet Explorer, Internet Information Server, DOS, Windows and Visual Basic are either registered trademarks or trademarks of Microsoft Corporation in the United States of America and/or other countries.

Netscape FastTrack and Navigator are trademarks of Netscape Communications Corporation.

Novell is a registered trademark, and NetWare is a trademark, of Novell Incorporated.

RoboHELP is a trademark of eHelp Corporation.

All other product or service names mentioned herein are trademarks or registered trademarks of their respective owners.

Copyright

Copyright © 2003 by Gupta Technologies LLC. All rights reserved.

SQLBase OLE DB Data Provider

20-6245-0001

July 2003

Contents

Preface.....	1-9
Audience.....	1-10
Notation conventions	1-10
Other helpful resources	1-11
Send comments to.....	1-11
1 Introduction to SQLBase Connectivity.	1-1
Gupta client and server technology.....	1-1
SQL.INI and the Connectivity Administrator	1-2
Running Connectivity Administrator	1-2
SQL.INI server-side example	1-9
SQL.INI client-side example.....	1-11
SQLBase native connectivity	1-11
2 ODBC Driver Introduction.....	2-1
How does it work?.....	2-2
ODBC overview.....	2-2
About the SQLBase ODBC driver.....	2-3
System requirements	2-3
Supported Features	2-3
ODBC Conformance Level.....	2-3
Number of Connections and Statements Supported.....	2-4
Isolation and Lock Levels Supported	2-4

Supported ODBC Functions	2-4
Deprecated functions	2-5
Data Types	2-6
Configuring Data Sources	2-7
Data Source Properties dialog	2-7
3 Making an ODBC Connection	3-1
Connection strings	3-1
Connection string keywords	3-1
Logon dialog	3-2
4 Introducing SQLBase OLE DB Data Provider4-1	4-1
Overview	4-2
OLE DB components	4-2
Recommended reading	4-2
About SQLBase OLE DB Data Provider	4-2
SQLBase OLE DB Data Provider name	4-4
Data source name	4-4
Supported programming tools	4-4
Installing	4-4
5 Using ADO	5-1
ADO Overview	5-2
ADO features	5-2
Remote Data Service	5-2
Programming with ADO	5-3
Threading	5-3
ADO Objects	5-3
Dynamic Properties Collections	5-4
Working with Data	5-5
SQLBase provider-specific dynamic properties	5-6
Sample applications	5-6
Connecting to data through the Provider	5-6

Establishing a connection using ADO	5-7
Coding Sample using ADO.	5-7
6 Supported OLE DB COM Objects	6-1
Supported COM objects	6-2
Data Source	6-3
Session	6-5
Command	6-6
Rowset	6-8
Error	6-9
7 ADO Mapping to Gupta	7-1
Isolation level mapping	7-2
ADO mapping to Gupta interfaces	7-2
Command object mapping	7-3
Connection object mapping.	7-3
Error object mapping.	7-4
Field object mapping.	7-5
Parameter object mapping	7-6
Recordset object mapping	7-7
8 .NET Data Provider.	8-1
SQLBase .NET requirements	8-1
SQLBase .NET implementation	8-1
Mapping SQLBase Datatypes to Common Language Runtime	
8-2	
SQLBaseConnection object	8-3
SQLBaseConnection properties	8-3
SQLBaseConnection methods	8-4
SQLBaseCommand object	8-6
SQLBaseCommand properties.	8-6
SQLBaseCommand methods.	8-6
SQLBaseDataReader object.	8-8
SQLBaseDataReader methods	8-8

Get... methods	8-8
Get methods for long (BLOB) data	8-8
GetDataTypeName.	8-9
GetFieldType	8-9
GetName	8-9
GetOrdinal	8-9
GetValues.	8-10
IsDBNull	8-10
NextResult	8-10
SQLBaseDataAdapter object	8-10
SQLBaseDataAdapterProperties	8-11
SQLBaseTransaction object	8-11
SQLBaseTransaction Properties	8-11
SQLBaseTransaction methods.	8-12
SQLBaseParameter object	8-12
Properties	8-12
SQLBaseParameters object	8-13
Add	8-13
SQLBaseException object	8-14
SQLBaseException Properties	8-14
SQLBase classes and transaction state	8-14
9 Introduction to the JDBC driver	9-1
Introduction	9-1
JDK Versions	9-1
Installation	9-2
Tested Platforms.	9-2
Getting Started	9-2
CLASSPATH	9-2
Server Configuration.	9-2
SQLBase Support Issues	9-3
SQL Operations Support	9-3
DDL Operations Support	9-3

DML Operations Support	9-3
Multiple Connection Support.	9-3
Multi-threading Support	9-3
Transaction Model	9-3
URL formats	9-4
URL keywords:	9-4
Data types and conversions	9-5
SQLBase Supported Data Types:	9-6
For READING Operations using ResultSet:	9-6
Bind variable support	9-7
Stored procedure support	9-7
Named Cursor Support	9-7
Error Message Support.	9-7
Application and Applet Development	9-8
Restrictions.	9-8
Glossary	Glossary-1
Index	Index-1

Preface

This manual provides reference information about using SQLBase OLE DB Data Provider to access SQLBase from visual programming tools, such as Microsoft Visual Basic, C++, and Delphi.

This preface provides the following information:

- Who should read this manual.
- The organization of this manual.
- The documentation format.
- The notation conventions used in this manual.
- Related publications.

Audience

This manual is intended for **Application Developers** building client applications that access Gupta SQLBase using frontend products such as Visual Basic, C++, and Delphi.

This manual assumes you have a working knowledge of SQLBase, relational databases in general, and SQL.

Notation conventions

Before you start using this manual, it is important to understand the typographical conventions we use in this manual:

Formatting Convention	Type of Information
You	A developer who reads this manual
User	The end-user of applications that you write
bold type	Menu items, push buttons, and field names. Things that you select. Keyboard keys that you press.
Courier 9	Development language code example
SQL.INI MAPDLL.EXE	Program names and file names
Precaution	Warning:
Vital information	Important:
Supplemental information	Note:
Alt+1	A plus sign between key names means to press and hold down the first key while you press the second key

Other helpful resources



Gupta Books Online. The Gupta document suite is available online. This document collection lets you perform full-text indexed searches across the entire document suite, navigate the table of contents using the expandable/collapsible browser, or print any chapter. Open the collection by selecting the Gupta Books Online icon from the **Start** menu or by double-clicking on the launcher icon in the program group.

World Wide Web. Gupta Technologies LLC's world wide Web site contains information about Gupta Technologies LLC's partners, products, sales, support, training, and users. The URL is <http://www.guptaworldwide.com>.

The technical services section of our Web site is a valuable resource for customers with technical support issues, and addresses a variety of topics and services, including technical support case status, commonly asked questions, access to Gupta's online newsgroups, links to shareware tools, product bulletins, white papers, and downloadable product updates.

Our Web site also includes information on training, including course descriptions, class schedules, and certified training partners.

Send comments to...

If you have any comments or suggestions regarding this manual, please send them to:

Technical Publications Department
Gupta Technologies
975 Island Drive
Redwood Shores, CA 94065

or send email, with comments or suggestions to:

techpubs@guptaworldwide.com

Chapter 1

Introduction to SQLBase Connectivity

Gupta client and server technology

Connectivity has two faces at Gupta Technologies. We make SQLBase, the relational database server discussed in this book. And we also make Team Developer, a set of tools for creating powerful client applications that connect to SQLBase and to many other data sources as well.

The details of client connectivity for applications written with Team Developer to any data source are documented in another Gupta book, *Connecting Gupta Objects to Databases*. In contrast, the book you are reading now will discuss how to connect any client application, written with any programming tool, to Gupta SQLBase.

But the database administrator still needs to be concerned with client connectivity, for two reasons:

- You need client tools for DBA tasks like configuration, backup, and testing. Since you are likely to use Gupta's tools for these purposes, you will still need to have some knowledge of client connectivity.
- Client computers, even those that do not use Gupta client tools, still need at least a rudimentary client-side SQL.INI file on their machines, for purposes of specifying basic server, database and protocol information.

SQL.INI and the Connectivity Administrator

Both Gupta's client tools and their server tools use a configuration file to store connectivity information. In versions of SQLBase prior to 8.5, this file was always named SQL.INI, and choices for its location were limited. In version 8.5 the file name and location are now under your control, although the default name is still SQL.INI, and that is the name we use to refer to the configuration file in Gupta documentation. The ability to have any name and location for the configuration file is a feature that supports multiple concurrent SQLBase installations on a single computer, also new in version 8.5.

For new SQLBase installations, SQL.INI is created during the install process and is tailored to your answers to the install questions. By default, it is found in the same directory as the SQLBase executables. This default SQL.INI file is functional immediately and, for many users, it is unnecessary to ever change SQL.INI until new servers or new databases are added to the installation. Experienced users may wish to change SQL.INI settings to achieve specific performance and communication goals.

The remaining chapters in this book will discuss how to configure SQLBases' drivers and data providers. In some cases such configuration involves changes to SQL.INI. Even in cases where SQL.INI is not directly involved, such as the OLE DB data provider, it is still indirectly involved since individual databases must be registered in SQL.INI before they are accessible to client applications.

Although SQL.INI can be modified with a text editor such as Notepad, uncontrolled manual changes can accidentally introduce problems, such as being unable to use a specific protocol against SQLBase or being unable to access a particular database. Gupta recommends that you use the Connectivity Administrator instead to make changes.

In this book, we discuss only basic SQL.INI concepts. The *Database Administrator's Guide* contains much more detailed information about every concept that is briefly discussed in this chapter.

Running Connectivity Administrator

Connectivity Administrator is one of the options in the Gupta program group.

Depending on what you have installed, the main window can show two tabs. The Connectivity tab works on client-side settings to add or remove, enable or disable routers and protocols. Choose the Server tab to add or remove databases and enable or disable protocols.

When configuring connectivity for a SQLBase server, you can:

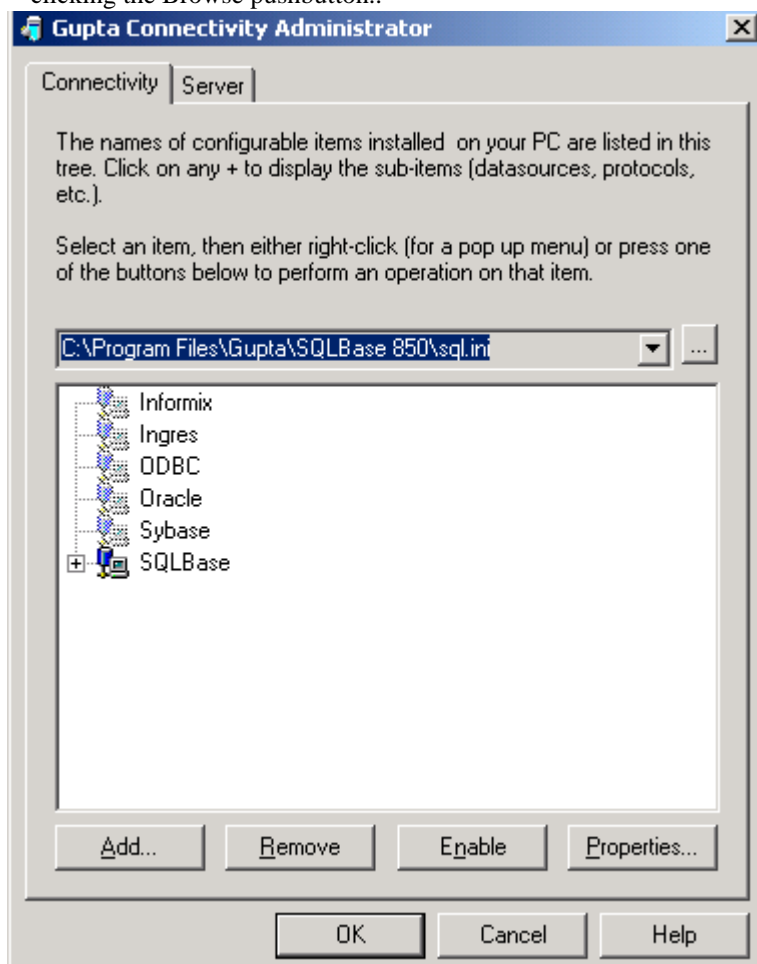
- Set up the options for a SQLBase server
- Enable and disable communication protocols

- Modify the listening protocols of a database name
- Add or remove a server name from a listening protocol
- Add or remove a database name
- View a list of database names
- Change the settings of an installed server

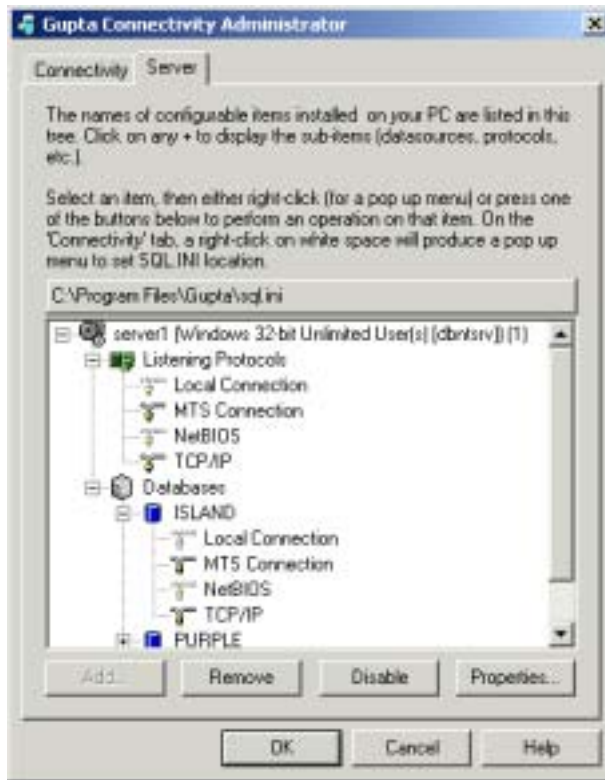
When configuring connectivity for a SQLBase client, you can:

- Edit the name, encryption level, and runtime directory of the client itself.
- Enable and disable listening protocols.
- Edit the properties of some listening protocols (TCP/IP and local connection)
- Associate new server names with a specific listening protocol, or remove server names from a listening protocol.
- Limit access to specific databases on a server, or access all databases.
- Change properties of a server/protocol combination, such as the TCP/IP listen port.
- Enable, disable, and configure data sources other than SQLBase. (That topic is outside the scope of this book; see *Connecting Gupta Objects to Databases*.)

The illustration below shows how the Client tab allows you to change the name and location of the configuration file you wish to edit, by directly typing the name or by clicking the Browse pushbutton..



On both tabs, you left-click to select an item, double-click to change properties, or right-click to display a menu of connectivity functions.



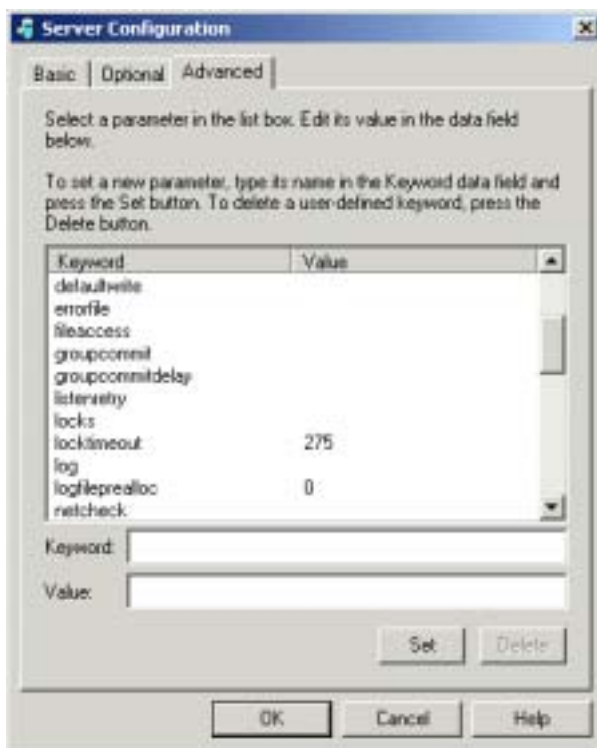
If you double-click on the server node (“server1” above), or right-click on it and choose Properties, you can work with the Server Configuration dialog box, which has three tabs:



The Basic tab controls the server name. The name you use in SQL.INI files on client computers must match the name found here.



The Optional tab allows you to configure other commonly-changed and critical settings.



The Advanced tab allows you to change any server keyword, presuming that you know the value that should be associated with the keyword. Detailed information about values and keywords is available in Chapter 3 of the *Database Administrator's Guide*.

You can also configure database settings, by double-clicking on a database node or by right-clicking it and then choosing Properties.



Database setting changes are limited to the database name and activating/deactivating listening protocols. Note that in the example above, three of the listening protocol checkboxes are disabled. That is because those protocols have been disabled at the server level, so they cannot be manipulated at the database level.

You can add a new database to the server by right-clicking the “Databases” node, then choosing “Add database”.

SQL.INI server-side example

Shown below is a portion of a typical SQL.INI file containing basic information about a SQLBase server and its databases. Some of the lines in the file are annotated to explain their purpose.

```
[dbntsrv]
```

“dbntsrv” is the abbreviation for the SQLBase unlimited-user version section. Your version might be different. For example, the 10-user version uses the abbreviation “dbnt10sv”.

```
servername=server1,sqlmpipe,sqlws32
```

The default name for a SQLBase server is “server1”. Following the server name are the names of the listening protocols that apply to the entire server. (As we will see later, individual databases need not implement all of these protocols.) “sqlmpipe” represents the MTS (COM+) listener, and “sqlws32” is the TCP/IP listener. These keywords are not case-sensitive.

```
dbname=ISLAND,SQLMPIPE,sqlws32
```

“ISLAND” is the demonstration/tutorial database installed with SQLBase. By default, it is assigned the same listening protocols as the server itself. Each database in a server must have a “dbname” line within the server section of SQL.INI.

```
cache=2000
sortcache=2000
readonly=0
oracleouterjoin=0
logfileprealloc=0
partitions=0
optimizerlevel=2
locktimeout=275
dbdir=C:\Program Files\Gupta
```

Databases are physical files, each in its own subdirectory with the same name as the database itself. By default, these subdirectories are presumed to be located under the SQLBase program directory, but you can change the path to point elsewhere.

```
dbname=SQLCON,sqlws32
```

Database “SQLCON” is used for the SQLConsole DBA utility. Note that in this example it implements only the TCP/IP listening protocol, not the MTS protocol.

```
dbname=PURPLE,sqlmpipe,sqlws32
```

Database “PURPLE” was added using Connectivity Administrator. Note that it implements both of the listening protocols used by the server itself.

```
[dbntsrv.dll]
comdll=sqlmpipe
comdll=sqlws32
```

Section “dbntsrv.dll” determines which listening protocols will be loaded when the SQLBase server starts.

```
[dbntsrv.ws32]
listenport=2155
```

Each listening protocol has a separate section in SQL.INI with optional extra information. The TCP/IP listener has a default port, 2155. You can override this, but if you do, you must insure that any SQL.INI files on client machines are altered to match the new port number.

Note: The default port for the MTS (COM+) listening protocol, SQLMPIPE, is 2156. This is not subject to overriding in SQL.INI - the port number is a runtime parameter that you supply to the SQLBase Resource Manager, SQLBrm.exe. For more information, see the SQLBase release notes. You can override this, but if you do, you must insure that any SQL.INI files on client machines are altered to match the new port number.

SQL.INI client-side example

This is a good spot to illustrate how SQL.INI files on client machines must match the SQL.INI on the server. Here are a few selected sections from SQL.INI on a client machine.

```
[win32client.dll]
comdll=sqlws32
```

The [winclient32.dll] section shows the communication libraries that the client loads when it first tries to connect to a database. “sqlws32”, as with the server, indicates the TCP/IP listening protocol. This client is going to take part in MTS transactions, so you might expect to see a “comdll=sqlmpipe” line in this section as we did when discussing the server. However, sqlmpipe is only used on the server. Clients use TCP/IP for both ordinary database communications and MTS transactions, so only the sqlws32 line is required in this case.

```
[win32client.ws32]
serverpath=server1,devsrvr,2156/*
```

The [win32client.ws32] section provides more detail about the TCP/IP listening protocol. In the “serverpath” line, we see the name of the server (“server1”), which must match the name used in the SQL.INI file on the server machine. Then we see the name of the computer that hosts the SQLBase server (“devsrvr”). Then we see the listening port (2156). In this example, the SQLBase server is running the SQLMPIPE protocol for MTS (COM+) compatibility. Therefore the client SQL.INI points to the SQLMPIPE port, 2156, rather than the default TCP/IP port of 2155. If you change this port number to 2155, the client will still be able to communicate with the SQLBase server using TCP/IP, but will be unable to participate in MTS transactions. In any case, the port numbers used by the client SQL.INI file must match those used in the server SQL.INI file. Finally, the “/*” following the port number means that all databases on server1 are visible to the client. If you specified only one database name, such as “2156/ISLAND”, then only that database would be visible to the client.

The “serverpath” line is the only way that the client can determine how to reach the server, so its presence is critical. There can be multiple “serverpath” lines in this section, each pointing to different servers or different databases.

SQLBase native connectivity

This book discusses industry-standard drivers and database providers for SQLBase. In addition, SQLBase has native connectivity that can be used easily by Gupta client tools, and can be called from other client tools through an API. For more information about this native protocol, read the *SQL Application Programming Interface* book.

Chapter 2

ODBC Driver Introduction

This chapter describes SQLBase's support for the Microsoft Open DataBase Connectivity (ODBC) standard. The ODBC standard is an application programming interface (API) specification written by Microsoft. It is an effort to standardize the way in which frontend products access database servers from different vendors.

How does it work?

Each relational database vendor has its own proprietary API. If you want your client application to access a database server, you must code to the database vendor's API.

Alternatively, you can use a router or gateway product to translate from one vendor's API to another vendor's API. Gupta's SQLRouter and SQLGateway products provide such a service. They sit between a Gupta SQLWindows or Report Builder client application and a non-SQLBase database server and translate Gupta API calls into non-SQLBase API calls. Gupta has SQLNetwork products that support DB2, DB2/400, Oracle, SQL Server, and more.

The ODBC standard takes another approach. It calls for all client applications to write to the ODBC standard API and for all database vendors to provide support for it. It then relies on third-party access tools or database drivers (such as the Gupta SQLBase ODBC driver) that conform to the ODBC specification to translate the ODBC standard API calls generated by the client application into the database vendor's proprietary API calls.

ODBC overview

The ODBC architecture specifies four components:

- The *application*. The application requests a connection with a data source. It calls ODBC functions to submit SQL statements and retrieve results.
- The *Driver Manager*. The Driver Manager is a dynamic-link library (DLL) that is provided by Microsoft and is included with the Gupta SQLBase ODBC driver. Its primary purpose is to load drivers on behalf of an application.
- The *driver*. To access SQLBase, you use the Gupta SQLBase ODBC driver, which is packaged with SQLBase.

The driver processes ODBC function calls, submits SQL requests to a specific data source, and returns results to the application. If necessary, the driver modifies an application's request so that the request conforms to syntax supported by the associated DBMS.

- The *data source*. This consists of the data you want to access and its associated DBMS (in this case, SQLBase), operating system, and network platform (if any).

For more detailed information on ODBC architecture and implementation, read the Microsoft's *Programmer's Reference* manual that accompanies the ODBC Software Developer's Kit (SDK).

About the SQLBase ODBC driver

The SQLBase ODBC driver supports the Gupta SQLBase database system in the following operating systems: Windows 98, ME, NT, 2000, Server 2003, and XP.

The driver file name is sqlbaseodbc.dll. Driver and data source configuration are done through sqlbaseodbcsetup.dll.

The information in this chapter is also available in an online help file that is accessible during ODBC data source configuration.

System requirements

The Gupta SQLBase ODBC driver (*SQLBaseODBC.dll*) requires:

- SQLBase version 8.5 or higher.
- One of the supported operating systems: Microsoft Windows 98, ME, NT, 2000, XP, or Server 2003.
- The Microsoft 3.5 Driver Manager. This is included with the Gupta SQLBase ODBC driver.

To communicate with the server, the SQLBase driver requires SQLWNTM.DLL, SQLBAPW.DLL, SQLBASEUTIL.DLL, and a communication DLL (for example, SQLWS32.DLL for TCP/IP). The directory containing these files must be on your path.

Client applications written with a Gupta programming tool, such as SQLWindows, cannot use ODBC to connect to SQLBase. For such applications, use the native SQLBase routers instead.

Supported Features

ODBC Conformance Level

The API functions supported are listed in “Supported ODBC Functions”. The driver conforms to ODBC Core and Level 1 completely. In addition, many Level 2 functions and some Level 3 functions are supported. When queried for API conformance, the driver responds that it is Level 2 compliant.

The driver supports the SQL Minimum Grammar as well as ODBC Escape Sequences for date, time and timestamp values, outer join and scalar functions.. The driver also supports backward and random fetching in SQLExtendedFetch and SQLFetchScroll.

Number of Connections and Statements Supported

The SQLBase database system supports multiple connections and multiple statements per connection.

Isolation and Lock Levels Supported

SQLBase supports the following standard ODBC isolation levels:

SQL_TXN_REPEATABLE_READ, SQL_TXN_READ_COMMITTED and SQL_TXN_READ_UNCOMMITTED.

SQLBase supports page-level locking.

Supported ODBC Functions

Gupta's SQLBase ODBC Driver supports the following functions:

- SQLAllocHandle
- SQLBindCol
- SQLBindParameter
- SQLBrowseConnect *
- SQLCancel
- SQLCloseCursor
- SQLColAttribute
- SQLColumnPrivileges
- SQLColumns
- SQLConnect
- SQLCopyDesc
- SQLDescribeCol
- SQLDescribeParam *
- SQLDisconnect
- SQLDriverConnect
- SQLEndTran
- SQLExecDirect
- SQLExecute
- SQLFetch
- SQLFetchScroll
- SQLFreeStmt
- SQLForeignKeys
- SQLFreeHandle
- SQLGetConnectAttr
- SQLGetCursorName
- SQLGetData
- SQLGetDescField
- SQLGetDescRec
- SQLGetDiagField
- SQLGetDiagRec
- SQLGetEnvAttr

SQLGetInfo
SQLGetStmtAttr
SQLGetTypeInfo
SQLMoreResults *
SQLNativeSql
SQLNumParams
SQLNumResultCols
SQLParamData
SQLPrepare
SQLPrimaryKeys
SQLProcedureColumns **
SQLProcedures
SQLPutData
SQLRowCount
SQLSetConnectAttr
SQLSetCursorName
SQLSetDescField
SQLSetDescRec
SQLSetEnvAttr
SQLSetPos
SQLSetStmtAttr
SQLSpecialColumns
SQLStatistics
SQLTablePrivileges
SQLTables

* Exported by the driver but currently not fully implemented. Will return standard ODBC error to indicate that it is not implemented.

** Seach patterns in SqlProcedureColumns are limited to the % symbol as the last character of the pattern. Any other wildcard characters will return a not-supported error.

Deprecated functions

Below is an alphabetic list of deprecated ODBC API functions. Note that these functions can still be used by an ODBC application, but they are implemented by the ODBC Driver Manager and not by the driver. The driver manager maps these deprecated functions into other functions that are actually implemented by the driver.

Deprecated function	Mapped to
SQLAllocConnect	SQLAllocHandle
SQLAllocEnv	SQLAllocHandle
SQLAllocStmt	SQLAllocHandle

Deprecated function	Mapped to
SQLError	SQLGetDiagRec
SQLExtendedFetch	SQLFetchScroll
SQLFreeConnect	SQLFreeHandle
SQLFreeEnv	SQLFreeHandle
SQLGetStmtOption	SQLGetStmtAttr
SQLParamOptions	SQLSetStmtAttr
SQLSetConnectOption	SQLSetConnectAttr
SQLSetParam	SQLBindParameter
SQLSetScrollOptions	SQLSetStmtAttr
SQLSetStmtOption	SQLSetStmtAttr
SQLTransact	SQLEndTran

Data Types

The SQLBase data types are mapped to the standard ODBC data types as follows:

SQLBase	ODBC
Char	SQL_VARCHAR
Date	SQL_TYPE_DATE
Decimal	SQL_DECIMAL
Double Precision	SQL_DOUBLE
Integer	SQL_INTEGER
Long Varchar	SQL_LONGVARCHAR
Number	SQL_DOUBLE
Real	SQL_REAL
Smallint	SQL_SMALLINT
Time	SQL_TYPE_TIME
Timestamp	SQL_TYPE_TIMESTAMP

SQLBase	ODBC
Varchar	SQL_VARCHAR

Configuring Data Sources

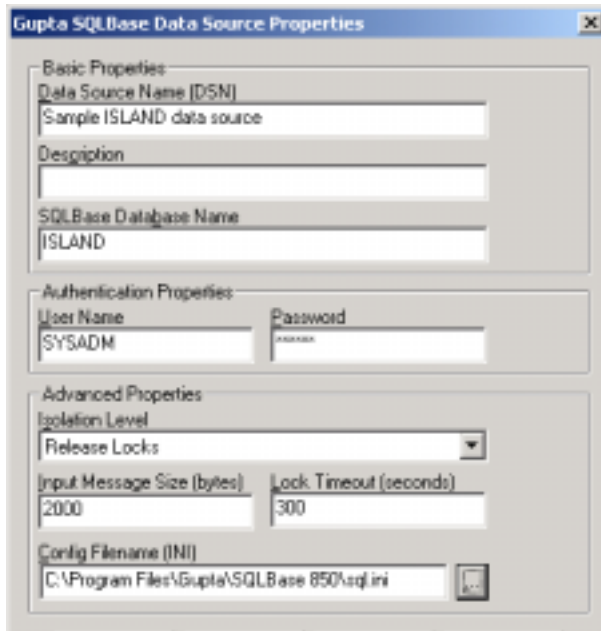
To configure a SQLBase data source, do the following:

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 SQLBase Data Source Properties dialog box.

If you are configuring a new data source, click Add to display a list of installed drivers. Select Gupta SQLBase and click Finish to display the SQLBase Data Source Properties dialog box.
3. Specify a data source name, a database name and optionally, a description.
4. Click Test Connect to attempt to connect to the data source using the connection properties specified in the Driver Setup dialog box. You may need to supply the name of the database server in the Test Connect dialog box.
5. If desired, configure optional data source settings in the Advanced section of the dialog.
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.

Data Source Properties dialog

Use the ODBC SQLBase Data Source Configuration dialog box to create new SQLBase data sources or configure existing data sources.



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

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

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

Isolation Level: The method of locking to use for this connection. Choices are Release Locks, Read Repeatability, Read Only, or Cursor Stability. For detailed information about how these levels work, read the SQLBase Language Reference manual, specifically the SET ISOLATION command.

Input Message Size: The number of bytes in the input message buffer. The default is determined by SQLBase. Increasing this value retrieves more records across the network in a single fetch.

Lock Timeout: The number of seconds SQLBase should wait for a lock to be freed before raising an error. Values can be -1 (wait forever) to 1800; the default is 300.

Config Filename (INI): The name (and, optionally, path) of the configuration file that contains information about SQLBase servers and databases. This name can be left blank; if so, the configuration file is searched for using the algorithm described in chapter 3 of the *SQLBase Database Administrator's Guide*.

Chapter 3

Making an ODBC Connection

Connection strings

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 of the `odbc.ini` file to use for the default connection information. Optionally, you may specify attribute or value pairs in the connection string to override the default values stored in the `odbc.ini` file.

You can specify either long or short attribute names in the connection string. Gupta recommends that you use long names. The connection string has the following format:

```
DSN=data source name[;attribute=value[;attribute=value]...]
```

An example of a connection string for SQLBase is:

```
DSN=SQLBASE_TABLES;SRVR=QESVR;DB=PAYROLL;  
UID=JOHN;PWD=XYZZY
```

Connection string keywords

Driver: Used in DSN-less connections, and is required to tell the Driver Manager which ODBC driver the rest of the connection string pertains to.

DataSourceName (DSN): A string that identifies a single connection to a SQLBase database system, for example, "Accounting".

Database (DB): The name of the database system to which you want to connect.

ServerName (SRVR): The name of the server that hosts the database to which you want to connect.

LogonID (UID): The default Logon ID used to connect to your SQLBase system. A Logon ID is required only if security is enabled on your database system. If so, contact your system administrator to get your Logon ID. This value is case-sensitive.

Password (PWD): Case-sensitive password.

IsoLevel (IL): A string that specifies how SQLBase will handle locking for this connection. Possible values are RL (Release Locks), RR (Read Repeatability), RO (Read-Only), and CS (Cursor Stability). For detailed information about how these levels work, read the *SQLBase Language Reference* manual, specifically the SET ISOLATION command.

InputMessageSize (IMS): Controls the number of bytes of the input message buffer. Increasing this value retrieves more records across the network in a single fetch.

LockTimeOut (LTO): Number of seconds SQLBase waits for a lock to be freed before raising an error. Values can be -1 to 1800. A value of -1 is infinite wait. Default is 300.

NoRecovery (NR): This keyword is accepted for backwards compatibility, but has no effect on the SQLBase server.

Logon dialog

Your application might supply enough information in the connection string to make the connection without prompting you. If not, when the SQLBase ODBC Driver needs additional information from you to make a connection, it displays a dialog box. Fill out the values in the dialog box as shown below:



1. Optionally, type the name of the server containing the SQLBase database tables you want to access or select the name from the Server Name drop-down list box, which displays the server names you specified in the Setup dialog box. Type Local to access a local SQLBase database.
2. Type the name of the database you want to access. If you specified a server name, you can select the name from the Database Name drop-down list box.
3. If required, type your user name.
4. If required, type your password.

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

Chapter 4

Introducing SQLBase OLE DB Data Provider

This chapter contains an overview of SQLBase OLE DB Data Provider functionality, software requirements, and installation information.

Overview

OLE DB is an application programming interface, written by Microsoft. Part of Microsoft's Universal Data Access strategy, it aims to facilitate access to information across data sources. The goal is to reliably offer high-performance data access that supports industry standards. The Universal Data Access strategy addresses the technical obstacles organizations face when trying to disseminate information across client/server networks and the World Wide Web. This task becomes more of a challenge given multiple platforms, legacy systems, and numerous programming languages and development tools. Based on open industry specifications, Universal Data Access allows developers to extend the capabilities of existing technologies.

OLE DB fits into this strategy by giving developers the ability to write applications that have uniform access to data regardless of the data source and development tool. Using OLE DB, it is possible for different pieces of an application to be written in different languages due to OLE DB's COM structure.

OLE DB components

The two main components of OLE DB are a data provider and a data consumer. An OLE DB data provider responds to OLE DB calls resulting in data being returned in a usable tabular format. A data consumer is an application or COM component that uses the OLE DB API to access a data source. Since OLE DB comprises a set of Component Object Model (COM) interfaces, it results in a high level of interoperability among programming tools. The COM interfaces support the full DBMS functionality of the data source. The COM specification allows you to use a variety of programming tools, such as Visual Basic and Delphi, to develop application components that can function within a single application.

Recommended reading

To make the best use of OLE DB, we recommend that you read *Microsoft OLE DB 2.0 Programming Reference and Data Access SDK*.

About SQLBase OLE DB Data Provider

Built upon OLE DB technology, SQLBase OLE DB Data Provider lets programmers access SQLBase functionality from a variety of supported visual programming tools. Historically, developers have had to call SQL/API functions or use the ODBC API to access a database. SQLBase OLE DB Data Provider exposes SQLBase Server properties more completely than the ODBC driver does, allowing developers to directly interface with the database and server using a variety of programming tools.

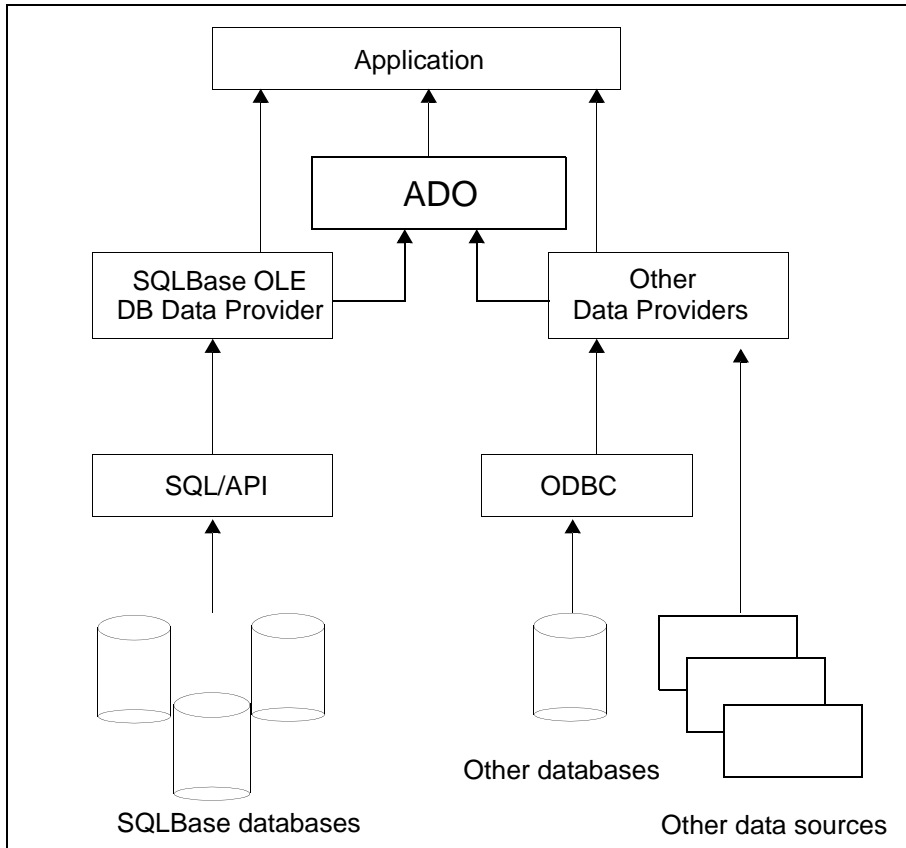
SQLBase OLE DB Data Provider is a set of COM objects that expose the SQLBase functionality needed by developers to build applications. In addition to standard OLE

DB conformance, the Provider interfaces enable access to DBMS functionality not normally provided by OLE DB, including backup, create database, and shutdown server.

A DLL, SQLBase OLE DB Data Provider runs in the address-space of the client program and is written over the SQL/API. Using OLE DB interfaces, the Provider lets you define DBMS components, including row containers, query processors, and transaction coordinators.

Another level of simplification: ADO. The Provider also provides high level access to your data through ActiveX Data Objects (ADO).

The following figure shows the role of SQLBase OLE DB Data Provider in the application development process.



SQLBase OLE DB Data Provider name

When you configure a programming tool to use the Provider, you may need to enter a provider name. For SQLBase OLE DB Data Provider, the name is:

SQLBaseOLEDB

The provider string is initialized for the open call that establishes a connection to SQLBase through the OLE DB provider. The name of the SQLBase OLE DB provider is hard-coded as a constant. The user ID, password, location and data source (tablename) are read from objects in your code. Here is a sample:

```
Dim sConnect As String      ' holds ADO connect string
' set up connection string and open connection to data
  source
sConnect = "Provider=SQLBASEOLEDB" _
  & ";Data Source=ISLAND" _
  & ";User ID=SYSADM" _
  & ";Password=sysadm"
```

For a more complete example of connecting to SQLBase using ADO, read *Chapter 5, Establishing a connection using ADO*.

Data source name

Some programming tools may prompt you for a data source name. For SQLBase OLE DB, the data source name is the name of the SQLBase database itself (for example, ISLAND).

Supported programming tools

SQLBase OLE DB Data Provider is designed to work with the following programming tools: Gupta Team Developer (SQLWindows), all the languages of Microsoft's Visual Studio, Delphi, Crystal Reports, PowerBuilder, and many others.

In general, SQLBase OLE DB Data Provider works with any development environment that supports OLE DB data access.

Installing

SQLBase OLE DB Data Provider is an option in the overall SQLBase installation. There is a checkbox for "OLE DB Data Provider" in the "Client Components" section of the installer..

Chapter 5

Using ADO

This chapter introduces you to the basics of ActiveX Data Objects (ADO) and how to use it.

The second section of this chapter includes a coding example that shows you how to use SQLBase OLE DB Data Provider and ADO.

To make the best use of ADO, we recommend you access the samples in:

www.microsoft.com/data

and access further information on ADO in:

www.microsoft.com/data/ado

ADO Overview

Microsoft ActiveX Data Objects (ADO) enables you to write an application to access and manipulate data in a database server through an OLE DB provider, such as the SQLBase provider. ADO's primary benefits are data source independence, high speed, ease of use, low memory overhead, and a small disk footprint. Since ADO is independent of the data source, you do not have to know all the intricacies of the database to take advantage of the provider.

For more general information about OLE DB providers, see the documentation for the Microsoft OLE DB SDK or visit the Microsoft OLE DB Web page. The SQLBase OLE DB Data Provider is a standard provider you can use with ADO as you would other standard providers.

ADO features

ADO provides a number of features for building client/server and Web-based applications. Using the SQLBase provider with ADO, you can take advantage of:

- Independently-created objects. Unlike Data Access Objects (DAO) or Remote Data Objects (RDO), you no longer have to navigate through a hierarchy to create objects because most ADO objects can be independently created. This allows you to create and track only the objects you need, and also results in fewer ADO objects and thus a smaller working set.
- Batch updating. If you are using client-side cursors, you can do batch updating. This helps improve performance by locally caching changes to data, then writing them all to the server in a single update.
- Support for stored procedures with input (not output) parameters.
- Different cursor types, including support of backend-specific cursors. With the SQLBase provider, when the user asks for the ability to fetch backward or scroll backward, the provider asks the server to enable its cursors.

Remote Data Service

ADO's Remote Data Service (RDS) allows data remoting, by which you can move data from a server to a client application or Web page, manipulate the data on the client, and return updates to the server in a single round trip. Previously released as Advanced Data Connector, RDS has been combined with the ADO programming model to simplify client-side data remoting. For more information, see the Remote Data Service documentation.

As part of the Microsoft Data Access Components, ADO and RDS are automatically installed and registered by a host product, such as Microsoft Internet Information

Server. The host product setup program may require that you specifically request the Microsoft Data Access Components in order to have ADO and RDS installed.

Programming with ADO

ADO is a dual-interface COM type library. The file name is msado15.dll. The program ID (ProgID) is "ADODB." In two- and three-tier database applications, ADO code that is to execute on the client uses the ProgID "ADOR" instead.

To use ADO with Microsoft Visual Basic or Microsoft Office, you also need to establish a reference to the ADO type library. From the **Project** menu, select **References**, check the box for "Microsoft ActiveX Data Objects 1.5 Library," and then click **OK**. ADO objects, methods, and properties will then be accessible through the VBA Object Browser and the IDE Editor.

To use ADO with Microsoft Visual J++, you also need to create a set of class files using the Java Type Library Wizard. From the **Tools** menu, select the Java Type Library Wizard, check the box for "Microsoft ActiveX Data Objects 1.5 Library," and then click **OK**. The wizard will then create the necessary Java class files for ADO.

For some programming languages, you also need to reference (using the #include statement) one or more additional files in your source code, as shown here:

C++	VBScript	JScript
adoint.h adoid.h	adovbs.inc	adojavas.inc

Threading

With SQLBase OLE DB Data Provider, use ADO as apartment-model threaded.

ADO Objects

Although you create ADO objects outside the scope of a hierarchy, the objects exist within hierarchical relationships, as shown in the ADO Object Model.

There are seven objects in the ADO object model:

Command — Maintains information about a command, such as a query string, parameter definitions, and so on. You can execute a command string on a Connection object or a query string as part of opening a Recordset object, without defining a Command object. The Command object is useful where you want to define query parameters, or execute a stored procedure that returns output parameters.

Connection — Maintains connection information with the data provider.

Error — Contains extended error information about an error condition raised by the provider. Because a single statement can generate two or more errors, the Errors collection can contain more than one Error object at a time, all of which result from the same incident.

Field — Contains information about a single column of data within a Recordset. The Recordset object features a Fields collection to contain all of its Field objects.

Parameter — A single parameter for a parameterized Command. The Command object features a Parameters collection to contain all of its Parameter objects.

Property — A provider-defined characteristic of an ADO object.

Recordset — A set of records returned from a query, and a cursor into those records. You can open a Recordset (that is, execute a query) without explicitly opening a Connection object. However, if you do first create a Connection object, you can open multiple Recordset objects on the same connection.

Cursor — In ADO, this is the equivalent of a recordset.

Each of these objects features a set of properties and methods with which you can manipulate the object and its contents.

Dynamic Properties Collections

The Connection, Command, and Recordset objects each support a Properties collection. The properties collection contains any dynamic (or "provider-specific") properties exposed through ADO by the SQLBase provider. You use the collection and the Item method to reference the property by its name or by its ordinal position in the collection, as shown:

```
Command.Properties.Item(0)
```

```
Command.Properties.Item("Name")
```

Because the Item method is a default method on an ADO collection, you can omit it:

```
Command.Properties(0)
```

```
Command.Properties("Name")
```

Further, the Properties collection itself is the default collection for the Connection, Command, and Recordset objects, so you can omit it as well:

```
Command(0)
```

```
Command("Name")
```

All of these syntax forms are identical. Which one you choose depends on your coding style preference.

Working with Data

In ADO, the Recordset object is the main interface to data. While the other objects are useful for managing connections, collecting error information, persisting queries, and so on, most of your code's interaction with ADO will involve one or more Recordset objects.

This Microsoft Visual Basic Scripting code generates a Recordset from a SQLBase data source:

```
set rstMain = CreateObject("ADODB.Recordset")
rstMain.Open "SELECT * FROM company", _
    "provider=SQLBaseOLEDB;Data source=ISLAND"
```

The first line creates the ADO Recordset object. The second line opens the recordset with a query. The third line specifies the SQLBase provider and the data source.

This generates a forward-only, read-only Recordset object. With a few modifications, you can obtain a more functional Recordset:

```
set rstMain = CreateObject("ADODB.Recordset")
rstMain.Open "SELECT * FROM company", _
    "provider=SQLBASEOLEDB;Data source=ISLAND",
    adOpenKeyset, adLockBatchOptimistic
```

This creates a fully scrollable and batch-updatable Recordset.

In order to reference ADO constants by name (instead of by value) when coding in a scripting language, as shown in these code examples, you must include the appropriate header file. For Microsoft Visual Basic Scripting code, include the file `adovbs.inc` in your code. For Microsoft JScript, include the file `adojavas.inc` in your code. You should always refer to constants by name rather than by value because the values may change from one version to the next.

However, because each provider is unique, how your application interacts with ADO will vary slightly between different providers. The differences you need to be aware of when using SQLBase OLE DB Data Provider are:

- Connection parameters in the `ConnectionString` property.
- Command object usage.
- Recordset behavior.

Note: For the Recordset object, Gupta recommends that you use server-side cursors (`adUseServer`). If client cursors are used, then at the end of your application's unit of work, all the changes should be committed using the `UpdateBatch()` method. `UpdateBatch()` throws up following warning message which can be ignored: "Run-time error '-2147217864 (80040e38)': Row cannot be located for updating. Some values may have been changed since it was last read"

SQLBase provider-specific dynamic properties

The Properties collections of the Connection, Command, and Recordset objects include dynamic properties specific to the SQLBase provider. These properties provide information about functionality beyond the built-in properties ADO supports.

After establishing the connection and creating these objects, use the Refresh method on the object's Properties collection to obtain the SQLBase provider-specific properties.

Sample applications

The SQLBase provider comes with a number of sample applications. These samples have been developed to show some of the ways ADO can be used with Visual BASIC to communicate with the SQLBase database product.

- simplefetch
Lets you connect to SQLBase and navigate, forward and backwards, through the COMPANY table - part of the ISLAND sample database for SQLBase. The code for this application is discussed in detail later in this chapter.
- schema
Lets you query the ADO schema types that the SQLBase OLE DB Data Provider supports.
- DBExplore
Lets you query SQLBase using SQL statements to return Recordsets from the SQLBase database. DBExplore.exe pulls out a Long Binary set of information, processes it, and writes it out to a file. Long text fields and Long Binaries are treated the same by the provider.
- SqlBaseNativeCmdSample
Demonstrates the SQLBase Native commands (load, unload, store, erase and retrieve) .

You must have installed Microsoft Visual Basic 6.0, the Microsoft Data Access components and the SQLBase OLE DB Data Provider in order to use these samples.

Connecting to data through the Provider

Whether or not you use ADO or direct OLE DB, you acquire database information using these steps:

1. Initialize the provider.
2. Establish a connection.

3. Open a recordset object.
4. Navigate through the database.
5. Process the recordset.
6. Disconnect.

Establishing a connection using ADO

ADO is an OLE DB consumer; it consumes OLE DB interfaces and data from the provider. To establish a connection to SQLBase through ADO, the sample we present in the next section follows these steps:

1. Create the various ADO objects.

For example, a connection object, a recordset object, and an error object.

2. Define a routine to output the Errors collection.
3. Define a routine to safely handle null fields.
4. Define a routine to fetch the values of the various fields from the database table.

Our sample assigns the field data to text boxes on the Visual BASIC form.

5. Define the code for the Connect button.

The name of the SQLBase OLE DB provider is `SQLBaseOLEDB`. This is hard-coded as a constant. The user, password, and data source (tablename) are read from `TextBox` objects.

6. Provide code for a button for closing the connection to SQLBase.
7. Provide code for a button that moves to the first record in the returned Recordset.
8. Provide code for a button that moves to the next record in the RecordSet.
9. Provide code for a button that moves to the previous record in the RecordSet.
10. Provide code for button that moves to the last record in the RecordSet.

Coding Sample using ADO

The following code sample illustrates how to use SQLBase OLE DB Data Provider. For more information, refer to the samples that ship with the product.

The code fragments show how to establish a connection, open a recordset object, and navigate through a SQLBase table using ADO in Visual BASIC. They are based on the `simplefetch` sample supplied with the OLE DB Provider. Portions of the code that don't specifically illustrate the concepts used have been excluded from this text for brevity.

1. Create the various ADO objects

```
Dim cnCompany As New ADODB.Connection
Dim rsCompany As New ADODB.Recordset
Dim erCompany As ADODB.Errors
```

2. Define two routines to output the Errors collection. OutputErrorMessage() is called whenever an error condition is signalled from the provider. This routine outputs the elements of the Errors collection, one at a time, by calling OutputMessage().

OutputMessage adds the messages to a ListBox on the form, and keeps the latest message in view at all times.

```
Private Sub OutputErrorMessage()

    ' loop through errors returned from provider, output to Messages
    Listbox

    Dim ErrLoop As Error

    Set erCompany = cnCompany.Errors
    For Each ErrLoop In erCompany

        OutputMessage ("Error: " _
            & Str(ErrLoop.NativeError) & " " _
            & ErrLoop.SQLState _
            & ErrLoop.Description)

    Next ErrLoop
End Sub

Private Sub OutputMessage(sMessage As String)
    lbMessages.AddItem (sMessage)
    lbMessages.TopIndex = lbMessages.ListCount - 1
End Sub
```

3. Define a routine to check for null fields. TextBox objects can be bound to your recordset object, to cause them to update and handle NULL values automatically. Because this sample obtains the values directly, IsNull is called on each value to prevent errors from Visual BASIC when returning NULL values.

```
Private Function GetField(sFieldName As String) As String
    If Not IsNull(rsCompany.Fields(sFieldName)) Then
        GetField = rsCompany.Fields(sFieldName)
    Else
        GetField = ""
    End If
```

End Function

4. Define a routine to fetch the values of the various fields from the database table, assigning them to TextBox objects on the form. A check for BOF or EOF prevents errors when attempting to display values outside the table. GetField() is used to prevent issues with NULL values returned from the provider.

```
Private Sub UpdateFieldsDisplay()
    If Not rsCompany.BOF And Not rsCompany.EOF Then
        ' fill fields of form from DB table
        CompanyID = GetField("Company_ID")
        CompanyName = GetField("Company_Name")
        Address = GetField("Address")
        City = GetField("City")
        State = GetField("State")
    End If
End Sub
```

5. Define the code for the Connect button. This is where the connection to the database is established, and the recordset object is connected to the COMPANY table.

```
Private Sub pbConnect_Click()
```

Establish error trapping, and clear the Errors collection to set up for the open attempt.

```
On Error GoTo FindErr
```

```
cnCompany.Errors.Clear      ' clear errors list
```

The provider string is initialized for the open call that establishes a connection to SQLBase through the OLE DB provider. The name of the SQLBase OLE DB provider is hard-coded as a constant. The user, password, and data source (tablename) are read from TextBox objects on the form.

```
Dim sConnect As String      ' holds ADO connect string

' set up connection string and open connection to data
source
sConnect = "Provider=SQLBaseOLEDB" _
    & ";Data Source=" & ebDataSourceName.Text _
    & ";User ID=" & ebUserID.Text _
    & ";Password=" & ebPassword.Text
```

```
cnCompany.Open sConnect
```

Connect the RecordSet object to the COMPANY table. The adOpenDynamic parameter is used to allow backward navigation through the table in this sample.

```
' open recordset on COMPANY table
rsCompany.Open "COMPANY", cnCompany, adOpenDynamic, ,
adCmdTable
```

Move the RecordSet object to the first record in the table, and call the routine to display the field values on the form.

```
' move to first record, update display of fields
rsCompany.MoveFirst
Call UpdateFieldsDisplay
Exit Sub
```

Provide an error routine that reports any problems with the connection to SQLBase or the COMPANY table.

```
FindErr:
' error occurred -- loop through error list and add messages
to listBox
Call OutputErrorMessages

' clear errors queue, restore arrow cursor, exit subroutine
cnCompany.Errors.Clear
Exit Sub

End Sub
```

6. Provide code for a button for closing the connection to SQLBase.

```
Private Sub pbDisconnect_Click()
    cnCompany.Close
End Sub
```

7. Provide code for a button that moves to the first record. We must call UpdateFieldsDisplay() to update the fields on each movement of the RecordSet object.

```
Private Sub pbFirst_Click()
```

```

        ' move to first record, update fields display
        rsCompany.MoveFirst
        Call UpdateFieldsDisplay
    End Sub

```

8. Provide code for a button that moves to the next record in the RecordSet. Special checks are made to stop at the end of the table.

```

Private Sub pbNext_Click()
    ' if we're not past last record, move to next record
    If Not rsCompany.EOF Then rsCompany.MoveNext

    ' if we're NOW past the last record...
    If rsCompany.EOF Then
        ' put cursor back on last record
        rsCompany.MoveLast
    Else
        Call UpdateFieldsDisplay
    End If
End Sub

```

9. Provide code for button that moves to the previous record in the RecordSet. Checks are made to stop at the beginning of the table.

```

Private Sub pbPrev_Click()
    ' if we're not before the first record, move to previous
    record
    If Not rsCompany.BOF Then rsCompany.MovePrevious

    ' if we're NOW before the first record...
    If rsCompany.BOF Then
        ' put cursor back on first record
        rsCompany.MoveFirst
    Else
        Call UpdateFieldsDisplay
    End If
End Sub

```

10. Provide code for button that moves to the last record in the RecordSet.

```

Private Sub pbLast_Click()
    ' move to last record, update fields display
    rsCompany.MoveLast
    Call UpdateFieldsDisplay
End Sub

```


Chapter 6

Supported OLE DB COM Objects

This chapter describes the OLE DB COM objects supported by SQLBase OLE DB Data Provider. In this chapter, the tables describe interfaces of objects the Provider supports, and whether our support is standard or specific.

Supported COM objects

SQLBase OLE DB Data Provider uses OLE DB COM objects and their supported interfaces to provide access to SQLBase functionality. SQLBase OLE DB Data Provider uses the following COM cotypes:

- **Data Source** (TDataSource) objects contain the machinery to connect to a data source, such as a file or a DBMS. These objects function as factory for sessions.
- **Session** (TSession) objects create a context for transactions and can be implicitly or explicitly transacted. A single data source object can create multiple sessions. Sessions are a factory for transactions, commands, and rowsets.
- **Command** (TCommand) objects create a text command, such as an SQL statement, that can query or update a data source. If the text command specifies a row set, such as an SQL Select statement, it creates a row set. A single session can create multiple commands.
- **Rowset** (TRowset) objects expose data from a data source in a tabular format. Row sets can be created from a session or a command. An index object is a special case of a row set object.
- **Error** (TError) objects are created when errors occur when accessing a data provider. They include status, return codes, and an optional custom error object. SQLBase OLE DB Data Provider utilizes the custom error object.

For a list and detailed description of the supported interfaces for the COM objects above, read Microsoft OLE DB documentation or visit the Microsoft Web site.

Data Source

SQLBase OLE DB Data Provider uses the following Data Source (TDataSource) object interfaces:

Mandatory	Optional
IDBCreateSession	IDBInfo
IDBInitialize	ISupportErrorInfo
IDBProperties	
IPersist	

Data Source properties

In addition to the properties inherited from its underlying classes, TDataSource has some specific properties of interest:

DBPROP_INIT_PROVIDERSTRING

This property can be used to supply up to three pieces of configuration information prior to making a connection to the database server. The parameter value, a string, must be formatted as follows:

```
INI=<ConfigFileName>;ims=<value>;oms=<value>
```

The configuration file name should include a fully qualified path. *ims*, input message buffer size, and *oms*, output message buffer size, can have a minimum value of 1 and a maximum value of 32000. (Default values are 2000 and 1000.) None of the three pieces of the string are required; any of them can be omitted. If there are two or more pieces in the string, the elements must be separated by semicolons.

Data Source descriptions

The following table briefly describes the data source interfaces and the level of support they receive.

Interface Name	Description	Support
IDBCreateSession	Calls on a data source object to obtain a new session.	Standard.
IDBInitialize	Initialize and uninitialize data source objects.	Standard, except enumerators are not supported.

Interface Name	Description	Support
<code>IDBProperties</code>	Sets and gets properties values for the data source object.	Standard, except enumerators are not supported.
<code>IPersist</code>	Returns the class identifier of an object.	Standard.
<code>IDBInfo</code>	Returns a list of provider-specific keywords, or information about literals.	Optional interface is supported.
<code>ISupportErrorInfo</code>	Shows whether an interface can return automated error messages.	SQLBase OLE DB COM objects that implement this interface also return error objects.

Session

SQLBase OLE DB Data Provider uses the following Session (TSession) object interfaces:

Mandatory	Optional
IGetDataSource IOpenRowset ISessionProperties	IDBCreateCommand IDBSchemaRowset ISupportErrorInfo ITableDefinition IIndexDefinition ITransactionLocal ITransactionObject ITransactionJoin

Session descriptions

The following table briefly describes the session interfaces and the level of support they receive.

Interface Name	Description	Support
IGetDataSource	IGetDataSource returns an interface pointer to the data source object.	Standard.
IOpenRowset	IOpenRowset opens and returns a rowset that includes all rows from a single base table or index.	Standard.
ISessionProperties	ISessionProperties returns information about the properties a session supports and the current settings of those properties. It is a mandatory interface on sessions.	Standard.
IDBCreateCommand	Consumers call IDBCreateCommand::CreateComm on a session to obtain a new command.	Standard.
IDBSchemaRowset	IDBSchemaRowset provides advanced schema information.	Standard.

Interface Name	Description	Support
ISupportErrorInfo	ISupportErrorInfo interface shows whether an interface can return automated error messages.	Standard.
ITableDefinition	The ITableDefinition interface exposes simple methods to create, drop, and alter data source tables.	Standard.
IIndexDefinition	The IIndexDefinition interface exposes simple methods to create, drop, and alter data source indexes.	Standard.
ITransactionLocal	ITransactionLocal is used to start, commit, and abort session transactions. It is optional	Standard.
ITransactionObject	ITransactionObject obtains the transaction object associated with a particular transaction level.	Standard.
ITransactionJoin	ITransactionJoin allows your session to enlist in a distributed transaction,	Standard

Command

SQLBase OLE DB Data Provider uses the following Command (TCommand) object interfaces:

Mandatory	Optional
IAccessor IColumnsInfo ICommand ICommandProperties ICommandText IConvertType	IColumnsRowset ICommandPrepare ICommandWithParameters ISupportErrorInfo

Command Class Interface descriptions

The following table briefly describes the command class interfaces and the level of support they receive.

Interface Name	Description	Support
IAccessor	The IAccessor describes how consumer buffer data is stored.	Standard.
IColumnsInfo	IColumnsInfo exposes column information for a rowset or prepare command. Also, it provides limited information for an array.	See Note.
ICommand	ICommand executes commands. This interface is mandatory on commands.	Standard.
ICommandProperties	ICommandProperties specifies properties from the Rowset property group to the command.	Standard.
ICommandText	When ICommandText is specified through SetCommandText, it replaces the existing command text.	Standard.
IConvertType	IConvertType provides command or rowset conversion type information. This interface is mandatory on commands, rowsets, and index rowsets.	Standard.
IColumnsRowSet	This interface supplies complete information about columns in a rowset.	Standard.
ICommandPrepare	This optional interface encapsulates command optimization, a separation of compile time and run time.	See Note.
ICommandWithParameters	This interface encapsulates parameters.	Standard.
ISupportErrorInfo	ISupportErrorInfo interface shows whether an interface can return automated error messages.	Standard.

Note: Because the SQLBase OLE DB provider exposes [ICommandPrepare](#), there are important stipulations to using the [IColumnsInfo](#) interface. Please see the OLE DB Programmer's Reference for further information.

Rowset

SQLBase OLE DB Data Provider uses the following Rowset (TRowset) object interfaces:

Mandatory	Optional
IAccessor IColumnsInfo IConvertType IRowset IRowsetInfo	IRowsetErrorInfo ISupportErrorInfo IRowsetChange

Rowset Interface descriptions

The following table briefly describes the rowset interfaces and the level of support they receive.

Interface Name	Description	Support
IAccessor	The IAccessor describes how consumer buffer data is stored.	Standard.
IColumnsInfo	IColumnsInfo exposes column information about rowset or prepared commands. It provides a limited set of array information.	See Note.
IConvertType	IConvertType provides command or rowset conversion type information. This interface is mandatory for commands, rowsets, and index rowsets.	Standard.
IRowset	IRowset is the base rowset interface. It provides methods for fetching rows sequentially, getting the data from those rows, and managing rows. IRowset requires IAccessor and IRowsetInfo.	Standard.
IRowsetInfo	IRowsetInfo provides information about a rowset. IRowsetInfo also provides methods for retrieving objects associated with the rowset. All rowsets must implement IRowsetInfo.	Standard.

Interface Name	Description	Support
IRowsetErrorInfo	IRowsetErrorInfo indicates whether a specific interface can return Automation error objects.	Standard.
ISupportErrorInfo	ISupportErrorInfo interface shows whether an interface can return automated error messages.	SQLBase OLE DB COM objects that implement this interface also return error objects.
IRowsetChange	IRowsetChange has methods for updating the columns of existing rows, deleting existing rows, and adding new rows.	Provider cannot update rowsets derived from multiple tables. Does not support updating, deleting or inserting rows for Schema Rowset. The updates are immediate.

Note: Because the SQLBase OLE DB provider exposes `ICommandPrepare`, there are important stipulations to using the `IColumnsInfo` interface. Please see the OLE DB Programmer's Reference for further information.

Error

SQLBase OLE DB Data Provider uses the following Error set (`TError`) object interfaces:

Mandatory	Optional
IErrorInfo IErrorRecords	None.

Error Interface descriptions

The following table briefly describes the error interfaces and the level of support they receive.

Interface Name	Description	Support
IErrorInfo	IErrorInfo returns error & return code information. It returns the error message, component name, and GUID of the interface where the error occurred, and the name and Help file topic that applies to the error. OLE DB error objects expose IErrorInfo at two levels.	Standard.
IErrorRecords	IErrorRecords is defined by OLE DB. It is used to add and retrieve records for an OLE DB error object. Information is passed to and from OLE DB error objects in an ERRORINFO structure.	Standard.
ISQLErrorInfo	ISQLErrorInfo returns the SQLSTATE & native error code.	Standard.

Chapter 7

ADO Mapping to Gupta

This chapter provides isolation level mapping to SQLBase.

This chapter also provides tables that list, for each ADO object:

- The ADO method, property, or collection.
- The OLE DB method called.
- Whether or not it is supported by the SQLBase OLE DB Data Provider.

Isolation level mapping

This table compares ADO isolation levels to SQLBase.

ADO	SQLBase
Default	CS (cursor stability)
adXactUnspecified	Set to default (CS)
adXactBrowse	RL (release lock)
adXactChaos	error
adXactCursorStability	CS (cursor stability)
adXactIsolated	error
adXactReadCommitted	CS (cursor stability)
adXactReadUncommitted	RL (release lock)
adXactRepeatableRead	RR (repeatable read)
adXactSerializable	error

ADO mapping to Gupta interfaces

There is a table following for each of these objects:

- Command
- Connection
- Error
- Field
- Parameter
- Recordset

The tables use this nomenclature for ADO methods, properties, and collections:

- Method()
- .Property
- Collection[]

Command object mapping

ADO Method, Property, or Collection	OLE DB Method(s) called	Support?
Execute()	ICommand::Execute	Yes
.ActiveConnection		Yes
.CommandText	ICommandText::SetCommandText	Yes
.CommandTimeout	ICommandProperties::SetCommandProperties (DBPROP_COMMANDTIMEOUT)	Yes
.CommandType		Yes
.Prepared	ICommandPrepare::Prepare ICommandPrepare::Unprepare	Yes Yes
Parameters[]	ICommandWithParameters::GetParameterInfo or DBSCHEMA_PROCEDURE_PARAMETERS ICommandWithParameters::SetParameterInfo	Yes Yes
Properties[]	IDBProperties::GetPropertyInfo ICommandProperties::GetProperties ICommandProperties::SetProperties	Yes Yes Yes

Connection object mapping

ADO Method, Property, or Collection	OLE DB Method(s) called	Support?
BeginTrans()	ITransactionLocal::StartTransaction	Yes
Close()		Yes
CommitTrans()	ITransactionLocal::Commit	Yes
Execute()	ICommand::Execute or IOpenRowset::OpenRowset	Yes Yes
Open()	IDBInitialize::Initialize IDBCreateSession::CreateSession	Yes Yes
RollBack()	ITransactionLocal::Abort	Yes

ADO Method, Property, or Collection	OLE DB Method(s) called	Support?
OpenSchema()	IDBSchemaRowset::GetRowset	Yes
.Attributes	ITransactionLocal::Abort	Yes
.CommandTimeout	ICommandProperties::SetCommandProperties (DBPROP_COMMANDTIMEOUT)	Yes
.ConnectionTimeout	IDBProperties::SetProperties (DBPROP_INIT_TIMEOUT)	Yes
.DefaultDatabase	IDBProperties::GetProperties (DBPROP_CURRENTCATALOG) IDBProperties::SetProperties (DBPROP_CURRENTCATALOG)	Yes Yes
.IsolationLevel	ITransactionLocal::StartTransaction	Yes
.Mode	IDBProperties::GetProperties (DBPROP_INIT_MODE) IDBProperties::SetProperties (DBPROP_INIT_MODE)	Yes Yes
.Provider	ISourcesRowset::GetSourcesRowset	Yes
.Version		Yes
Errors[]	IErrorRecords	Yes
Properties[]	IDBProperties::GetPropertyInfo IDBProperties::GetProperties IDBProperties::SetProperties	Yes Yes Yes

Error object mapping

ADO Method, Property, or Collection	OLE DB Method(s) called	Support?
.Description	IErrorRecords::GetErrorInfo	Yes
.HelpContent	IErrorRecords::GetErrorInfo	Yes
.HelpFile	IErrorRecords::GetErrorInfo	Yes
.NativeError	IErrorRecords::GetCustomErrorObject ISQLError::GetSQLInfo	Yes Yes

ADO Method, Property, or Collection	OLE DB Method(s) called	Support?
.Number		Yes
.Source	LErrorRecords::GetErrorInfo	Yes
.SQLState	LErrorRecords::GetCustomErrorObject ISQLError::GetSQLInfo	Yes Yes

Field object mapping

ADO Method, Property, or Collection	OLE DB Method(s) called	Support?
AppendChunk()	ISequentialStream::Write	Yes
GetChunk()	ISequentialStream::Read	Yes
.ActualSize	IAccessor::CreateAccessor IRowset::GetData	Yes Yes
.Attributes	IColumnInfo::GetColumnInfo	Yes
.DefinedSize	IColumnInfo::GetColumnInfo	Yes
.Name	IColumnInfo::GetColumnInfo	Yes
.NumericScale	IColumnInfo::GetColumnInfo	Yes
.OriginalValue	IRowsetUpdate::GetOriginalData	No
.Precision	IColumnInfo::GetColumnInfo	Yes
.Type	IColumnInfo::GetColumnInfo	Yes
.UnderlyingValue	IRowsetRefresh::GetLastVisibleData or IRowsetRefresh::GetVisibleData	No No
.Value	IAccessor::CreateAccessor IRowset::GetData IRowset::SetData	Yes Yes Yes
Properties[]	IColumnsRowset::GetAvailableColumns IColumnsRowset::GetColumnsRowset	Yes Yes

Parameter object mapping

ADO Method, Property, or Collection	OLE DB Method(s) called	Support?
AppendChunk()		Yes
.Attributes	ICommandWithParameters::GetParameterInfo or DBSCHEMA_PROCEDURE_PARAMETERS ICommandWithParameters::SetParameterInfo	Yes (No) Yes
.Direction	ICommandWithParameters::GetParameterInfo or DBSCHEMA_PROCEDURE_PARAMETERS ICommandWithParameters::SetParameterInfo	Yes (No) Yes
.Name	ICommandWithParameters::GetParameterInfo or DBSCHEMA_PROCEDURE_PARAMETERS	Yes (No)
.NumericScale	ICommandWithParameters::GetParameterInfo or DBSCHEMA_PROCEDURE_PARAMETER ICommandWithParameters::SetParameterInfo	Yes (No) Yes
.Precision	ICommandWithParameters::GetParameterInfo or DBSCHEMA_PROCEDURE_PARAMETERS ICommandWithParameters::SetParameterInfo	Yes (No) Yes
.Size	ICommandWithParameters::GetParameterInfo or DBSCHEMA_PROCEDURE_PARAMETERS ICommandWithParameters::SetParameterInfo	Yes (No) Yes
.Type	ICommandWithParameters::GetParameterInfo or DBSCHEMA_PROCEDURE_PARAMETERS ICommandWithParameters::SetParameterInfo	Yes (No) Yes
.Value	IAccessor::CreateAccessor ICommand::Execute	Yes Yes
Properties[]		Yes

Recordset object mapping

ADO Method, Property, or Collection	OLE DB Method(s) called	Support?
AddNew()	IRowsetChange::InsertRow	Yes
CancelBatch()	IRowsetUpdate::Undo	No
CancelUpdate()		Yes
Clone()	IRowsetLocate	No
Close()	IAccessor::ReleaseAccessor IRowset::ReleaseRows	Yes Yes
Delete()	IRowsetChange::DeleteRows	Yes
GetRows()	IAccessor::CreateAccessor IRowsetLocate::GetRowsAt IRowset::GetNextRows IRowset::GetData	Yes No Yes Yes
Move()	IRowsetLocate::GetRowsAt or IRowset::GetNextRows	No Yes
MoveFirst()	IRowsetLocate::GetRowsAt or IRowset::RestartPosition	No Yes
MoveLast()	IRowsetLocate::GetRowsAt	No
MoveNext()	IRowsetLocate::GetRowsAt or IRowset::GetNextRows	No Yes
MovePrevious()	IRowsetLocate::GetRowsAt or IRowset::GetNextRows	No Yes
NextRecordset()	IMultipleResults::GetResult	No
Open()	IOpenRowset::OpenRowset or ICommand::Execute	Yes Yes
Requery()	IOpenRowset::OpenRowset or ICommand::Execute	Yes Yes
Resync()	IRowsetRefresh::RefreshVisibleData	No
Supports()	IRowsetInfo::GetProperties	Yes

ADO Method, Property, or Collection	OLE DB Method(s) called	Support?
Update()	IRowsetChange::SetData and/or IRowsetUpdate::Update	Yes No
UpdateBatch()	IRowsetUpdate::Update	No
.AbsolutePage	IRowsetScroll::GetApproximatePosition IRowsetScroll::GetRowsAtRatio or IRowsetLocate::GetRowsAt	No No No
.AbsolutePosition	IRowsetScroll::GetApproximatePosition IRowsetScroll::GetRowsAtRatio or IRowsetLocate::GetRowsAt	No No No
.ActiveConnection	IDBInitialize::Initialize IDBCreateSession::CreateSession	Yes Yes
.BOF		Yes
.Bookmark	IAccessor::CreateAccessor IRowsetLocate::GetRowsAt	Yes No
.CacheSize	IRowsetLocate::GetRowsAt or IRowset::GetNextRows	No Yes
.CursorType	ICommandProperties::SetProperties	Yes
.EditMode	IRowsetUpdate::GetRowsStatus	No
.EOF		Yes
.Filter	IRowsetUpdate::GetPendingRows IRowsetLocate::GetRowsByBookmark or IRowsetView::CreateView IRowsetView::OpenViewChapter IViewFilter::SetFilter	No No No No No
.LockType	ICommandProperties::SetProperties	Yes
.MaxRecords	IOpenRowset::OpenRowset (DBPROP_MAXROWS) or ICommandProperties::SetCommandProperties (DBPROP_MAXROWS)	Yes Yes
.PageCount	IRowsetScroll::GetApproximatePosition	No
.PageSize	(PageCount, AbsolutePage properties)	No
.RecordCount	IRowsetScroll::GetApproximatePosition	No

ADO Method, Property, or Collection	OLE DB Method(s) called	Support?
.Source		Yes
.Status	IRowsetUpdate::GetRowStatus	No
Fields[]	IColumnInfo::GetColumnInfo	Yes
Properties[]	IRowsetInfo::GetPropertyInfo IRowsetInfo::GetProperties IRowsetInfo::SetProperties	Yes Yes Yes

Chapter 8

.NET Data Provider

SQLBase .NET requirements

You must have the .NET Framework already installed on your machine before the SQLBase .NET Data Provider will install correctly. Gupta recommends that you install the most recent service pack for the .NET Framework first.

The SQLBase installation program offers you the option of installing the .NET Data Provider. You must choose this option to use the provider. The actual component name is Gupta.SQLBase.Data.DLL. It is registered as a .NET component during installation

Any programming language that uses the provider will need to reference that component and its related namespace. For example, in Visual Basic.NET, you must choose menu item Project, Add Reference, locate the "Gupta SQLBase .NET Data Provider" item in the list on the ".NET" tab of the dialog, select it, and click OK.

To reference the namespace, you must add a declaration to your code. In Visual Basic.NET, for example, you might do this with a line above your main code module:

```
Imports Gupta.SQLBase.Data
```

SQLBase .NET implementation

The current version of SQLBase provides objects that implement eight of the .NET interfaces. The vast majority of their functionality is identical to that described in the [.NET Framework documentation](#) for the class library. In this chapter, we will only

document a property or method when there is something distinct about the way it is implemented in SQLBase.

SQLBaseConnection: implements the `IDbConnection` interface, establishes a connection to a specific data source.

SQLBaseCommand: implements the `IDbCommand` interface, executes a command against a data source, exposes parameters and can execute within the scope of a Transaction.

SQLBaseDataReader: implements the `IDataReader` interface and reads a scrollable, read-only stream of data from a data source.

SQLBaseDataAdapter: implements the `IDataAdapter` interface; populates a `DataSet` and resolves updates with the data source.

SQLBaseTransaction: implements the `IDbTransaction` interface and represents a transaction to be performed at a data source.

SQLBaseParameter: implements the `IDataParameter` interface, represents a parameter which can be passed to a SQL command.

SQLBaseParameters: implements the `IDataParameters` interface, which is a simple collection of `SQLBaseParameter` objects.

In addition to these eight objects, SQLBase provides **SQLBaseException**, a public interface/object which is used for throwing exceptions to a client in case of a SQLBase error during a client method call.

These objects are inherited from standard classes in the Microsoft .NET Framework. This chapter does not document the methods and properties that are inherited from those standard classes; it focuses only on the methods and properties that are specific to Gupta's implementation of the framework.

Mapping SQLBase Datatypes to Common Language Runtime

SQLBase Type	CLR Type
Integer	Int32
Smallint	Int16
Float	Single
Number	Decimal
Double	Double
Decimal	Decimal

SQLBase Type	CLR Type
Date	DateTime
Time	DateTime
Timestamp	DateTime
Char	String
Varchar	String
Graphic	byte[]
Vargraphic	byte[]
Binary	byte[]
Varbinary	byte[]
Longvar	byte[]
Longvargraphic	byte[]
Longbinary	byte[]
Char>254	char[]
Varchar>254	char[]

SQLBaseConnection object

For information about the methods, properties and events inherited from IDbConnection see the [.NET Framework documentation](#).

SQLBaseConnection properties

ConnectionString

This property supplies basic information about the data source. It can contain multiple keyword=value parameters separated by semicolons, as described below:

```
datasource|data source|ds|database|db=data source name string;
userid|user id|uid=user id string;
password|pwd=password string;
poolsize|pool size=integer;
connection lifetime=integer;)
```

All keywords except the first one are optional. Keywords are not case-sensitive. In cases where more than one keyword has the same meaning, the keywords are shown separated by pipe symbols in the diagram above.

SQLBaseConnection methods

BeginTransaction

Syntax `IDbTransaction BeginTransaction (IsolationLevel)`

Description Start a new transaction.

Parameters

IsolationLevel string. SQLBase supports the following values in the IsolationLevel parameter: ReadUncommitted (SQLBase Release Lock); ReadCommitted (SQLBase Cursor Stability); RepeatableRead (SQLBase Repeatable Read); Unspecified (SQLBase Repeatable Read).

Use of other unsupported isolation level values, such as Chaos or Serializable, will cause a SQLBase exception with "Unsupported isolation level : xxx." message.

ChangeDatabase

Syntax `void ChangeDatabase (string dbname)`

Description This method is not supported by SQLBase. Calling this method will throw a NotSupportedException exception to the caller.

Close

Syntax `void Close()`

Description This method logically closes your database connection. The connection may be cached rather than physically closed; for details on this, read *Connection pooling in the Open method* on page 8-5.

CreateCommand

Syntax `IDbCommand CreateCommand()`

Description Initialize a new SQLBaseCommand object.

GetSQLBaseSchemaTable

Syntax `DataTable GetSQLBaseSchemaTable (schemaGuid, restrictions)`

Description	<p>Return a DataTable object containing schema information for the currently connected database.</p> <p>SQLBase supports the following types of schema information:</p> <pre> SQLBaseSchemaGuid.Tables SQLBaseSchemaGuid.Columns SQLBaseSchemaGuid.Tables_Info SQLBaseSchemaGuid.Provider_Type SQLBaseSchemaGuid.Catalogs SQLBaseSchemaGuid.Foreign_Keys SQLBaseSchemaGuid.Primary_Keys SQLBaseSchemaGuid.Indexes SQLBaseSchemaGuid.Procedures SQLBaseSchemaGuid.Views SQLBaseSchemaGuid.Column_Privileges SQLBaseSchemaGuid.Table_Privileges </pre>
-------------	--

Parameters	<table> <tr> <td>schemaGuid</td><td>instance of class SQLBaseSchemaGuid</td></tr> <tr> <td>restrictions</td><td>string</td></tr> </table>	schemaGuid	instance of class SQLBaseSchemaGuid	restrictions	string
schemaGuid	instance of class SQLBaseSchemaGuid				
restrictions	string				

Open

Syntax	void Open(connection_string)
--------	---------------------------------------

Description	<p>Opens a database connection. When you call this method, SQLBase will attempt to reuse an existing open database connection from the connection pool. If this is not possible, it will create a new database connection.</p>
-------------	--

Parameters	<table> <tr> <td>connection_string</td><td>string. See the detailed description of this parameter in property ConnectionString, above.</td></tr> </table>	connection_string	string. See the detailed description of this parameter in property ConnectionString, above.
connection_string	string. See the detailed description of this parameter in property ConnectionString, above.		

Connection pooling in the Open method

SQLBase provides two of the keywords in the connection string parameter of the Connect method to assist with connection pooling. *poolsize* indicates how many SQLBase connections should be kept open, whether they are in immediate use or not. The default value is 5. *connection lifetime* indicates how long each connection should be maintained, in seconds. The default value is 60. These defaults are stored in the Windows registry under key
HKEY_LOCAL_MACHINE\Software\Gupta\SQLBase\DotNetDataProvider.

Each connection pool is specific to the application that calls the `Connect` method. When the number of active connections is still below or at the *poolsize* limit, connections that have been closed by the application remain cached (physically open and available for new connection requests) until their lifetime is reached, at which time they are physically disconnected. Each call to `Connect` resets the current values for *poolsize* and *connection lifetime*. Gupta recommends that you avoid issuing `Connect` calls in which the new value for *poolsize* is smaller than the one currently being used by the application.

SQLBaseCommand object

For information about the methods, properties and events inherited from `IDbCommand`, see the [.NET Framework documentation](#).

The `SQLBaseCommand` object is often initialized from the `CreateCommand` method of the `SQLBaseConnection` class. To release that class' underlying database cursor, it is important to call the `Dispose` method of `SQLBaseCommand` when your processing is finished. Otherwise the command object is released either when the connection is released or the garbage collector releases the object, and both of these will be quite late in most circumstances. In addition, relying upon the garbage collector may mean incomplete release of all resources connected to the command object, raising the possibility of eventually running out of resources in an application with heavy use of `SQLBaseCommand` objects.

SQLBaseCommand properties

CommandText

You should set the value of the `CommandText` property of the object before any other methods are called. For a Text Command type (see `CommandType` property), the `CommandText` property should contain a valid SQL statement. For a Stored Procedure type, the `CommandText` property should contain a procedure name. For a Table Direct type, the `CommandText` property should contain a table name.

CommandType

`SQLBase` supports three types of command objects: `TextCommand`, `StoredProcedure`, and `TableDirect`. These three The default type of each command object, upon creation, is Text Command. You are responsible for changing this type, if necessary, by altering the `CommandType` property of the object.

SQLBaseCommand methods

Cancel

Syntax `void Cancel ()`

Description Attempts to cancel the execution of a command. This method is not supported in SQLBase, and will throw a `NotSupportedException` when called.

GetSQLBaseParameter

Syntax `void GetSQLBaseParameter (parametertype, value, length)`

Description Retrieves the value of an internal SQLBase parameter.

`SetSQLBaseParameter` and `GetSQLBaseParameter` are not part of the `IDbCommand` interface. They manipulate the many SQLBase internal parameters. Detailed documentation for these parameters is available in the *SQLBase SQL Application Programming Interface Reference*, under the function *sqlset*.

Parameters

<code>parametertype</code>	string. One of the many parameter names defined in the SQLBase API Reference.
<code>value</code>	string, pass by reference. Contains the value returned from SQLBase.
<code>length</code>	integer, pass by reference. Buffer length of <i>value</i> .

SetSQLBaseParameter

Syntax `void SetSQLBaseParameter (parametertype, value, length)`

Description Attempts to set a SQLBase parameter value

`SetSQLBaseParameter` and `GetSQLBaseParameter` are not part of the `IDbCommand` interface. They manipulate the many SQLBase internal parameters. Detailed documentation for these parameters is available in the *SQLBase SQL Application Programming Interface Reference*, under the function *sqlset*.

Note: This method should be used only by advanced SQLBase developers. Calling this method while in the middle of a transaction can potentially cause unpredictable results in the transaction or the database.

Parameters

<code>parametertype</code>	string. One of the parameter names documented in the SQLBase API Reference.
<code>value</code>	string. The value to be assigned to the parameter.
<code>length</code>	integer. The length in bytes of <i>value</i> .

SQLBaseDataReader object

For information about the methods, properties and events inherited from `IDataReader`, see the [.NET Framework documentation](#).

This object provides the capability of reading a scrolling result set obtained by executing a command against a data source. `SQLBase` supports only a single result set. Datatypes are mapped to CLR-compatible types as described in *Mapping SQLBase Datatypes to Common Language Runtime* on page 8-2.

To make use of scrolling result sets, property `SQLBaseCommand.ResultSetMode` must be set to `TRUE`. The `SQLBaseDataReader` object implements interface `IEnumerable`, with a single method, `GetEnumerator`, which returns an `IEnumerator` object. That object has methods `MoveNext` and `Reset`, and get/set property `RowPos` (current row of result set), which allow you to move to a specific row in the result set.

SQLBaseDataReader methods

Get... methods

Syntax

```
Int16 GetInt16 (index)
Int32 GetInt32 (index)
Int64 GetInt64 (index)
Single GetFloat (index)
byte GetByte (index)
char GetChar (index)
Double GetDouble (index)
Decimal GetDecimal (index)
Guid GetGuid (index)
DateTime GetDateTime (index)
TimeSpan GetTime (index)
Boolean GetBoolean (index)
String GetString (index)
Object GetValue (index)
```

Description

Each of these methods retrieves a single value from a column in the dataset. The specific method chosen is dependent on the datatype of the column. If parameter *index* is not valid, exception `IndexOutOfRangeException` is thrown.

Parameters

`index` `int`. The position of the column in the result set.

Get methods for long (BLOB) data

Syntax

```
long GetBytes (index, offset, buffer, readBytes)
```

	long GetChars (index, offset, buffer, readBytes)	
Description	Reads bytes or characters from a column with a long datatype. If <i>index</i> is not found, throws exception <i>IndexOutOfRangeException</i> . Returns the number of bytes actually read, or, if <i>buffer</i> is null, the length of the column.	
Parameters	index	Long. Relative position of the column in the row.
	offset	long. Offset within the long data column at which reading is to begin. Zero-based.
	buffer	byte[] or char[], depending on method. Buffer to hold the results of the read.
	readBytes	long. Number of bytes to read into buffer.

GetDataTypeName

Syntax	string GetDataTypeName (index)	
Description	Get the SQLBase (not CLR) datatype for the specified column. If <i>index</i> is not found, throws exception <i>IndexOutOfRangeException</i> .	
Parameters	index	int. Position of column in the result set.

GetFieldType

Syntax	Type GetFieldType (index)	
Description	Return the .NET system type for the specified column. If <i>index</i> is not found, throws exception <i>IndexOutOfRangeException</i> .	
Parameters	index	int. Position of column in the result set.

GetName

Syntax	string GetName (index)	
Description	Get the column (field) name for the specified column. If <i>index</i> is not found, throws exception <i>IndexOutOfRangeException</i> .	
Parameters	index	int. Position of column in the result set.

GetOrdinal

Syntax	Int16 GetOrdinal (column)	
Description	Returns the ordinal (position in the row, beginning with one) of the column named in parameter <i>column</i> . If the column name is not found, throws exception <i>IndexOutOfRangeException</i> .	

Parameters	column	int. The position of the column in the result set.
------------	--------	--

GetValues

Syntax	<code>int GetValues (objectArray)</code>
--------	--

Description	Retrieves multiple values into an array of objects, rather than retrieving a single value as the other Get... methods do. It returns the actual number of values that were retrieved:
-------------	---

Parameters	objectArray	Object. An array of type Object that will receive values for each column in the result set. If there are fewer elements in the array than there are columns in the result set, then the method will fill only those elements, then stop.
------------	-------------	--

IsDBNull

Syntax	<code>bool IsDBNull (index)</code>
--------	------------------------------------

Description	Checks the specified column for a null value, and returns TRUE if found, or FALSE if not found. If <i>index</i> is not found, throws exception <i>IndexOutOfRangeException</i> .
-------------	--

Parameters	index	offset of column
------------	-------	------------------

NextResult

Syntax	<code>bool NextResult ()</code>
--------	----------------------------------

Description	Moves to the next result set. <i>SQLBase</i> supports only a single result set, so calls to this method always return FALSE.
-------------	--

SQLBaseDataAdapter object

This class does not inherit from *SqlDataAdapter*, since that class contains logic that is specific to Microsoft SQL Server. However, its function is quite similar; a data adapter has a database connection and a group of commands that are designed to move data from the database into a data set object.

This class inherits from the more general *DbDataAdapter* class. For information about the methods, properties and events inherited from *DbDataAdapter*, see the [.NET Framework documentation](#).

The methods of this class are all inherited from the standard *DbDataAdapter* class, which can be found in the Microsoft .NET Framework documentation. To complete the functionality of this class, *SQLBase* must implement four properties that are found in the *DbDataAdapterInterface*. Each of these properties has Get and Set methods.

SQLBaseDataAdapterProperties

DeleteCommand

Contains the text of a SQL query used to delete records.

InsertCommand

Contains the text of a SQL query used to insert new records.

SelectCommand

Contains the text of a SQL query used to select records.

When a .NET client calls the `dataAdapter.Fill ()` method it will execute the command which is contained in the `SelectCommand` property. The command object will return a `IDataReader` interface to the data adapter after executing the command , and the data adapter will use that interface to get data from the data source and fill the `System.DataSet` object. After that client can manipulate the data in the `DataSet` object through `DataSet` methods.

UpdateCommand

Contains the text of a SQL query used to update records.

When a .NET client calls the `dataAdapter.Update ()` method it will execute the three commands contained in the `UpdateCommand`, `InsertCommand` and `DeleteCommand` properties, and update the data source through these commands with the changes in the `DataSet`.

SQLBaseTransaction object

For information about the methods, properties and events inherited from `IDbTransaction`, see the [.NET Framework documentation](#).

SQLBaseTransaction Properties

Connection

Datatype: `SQLBaseConnection`. The `Connection` object which should be passed to this object in the constructor.

IsolationLevel

Datatype: IsolationLevel. Get only. The IsolationLevel enumeration which should be passed to this object in the constructor from the connection's BeginTransaction () method.

SQLBaseTransaction methods

Commit

Syntax `void Commit ()`

Description Commit the database transaction. The current transaction is complete. The client should call BeginTransaction method from the Connection object to start a transaction again.

Rollback

Syntax `void Rollback ()`

Description Roll back a transaction from a pending state.

SQLBaseParameter object

For information about the methods, properties and events inherited from IDbTransaction, see the [.NET Framework documentation](#).

This object has four properties and no methods. You must set its properties to appropriate values before supplying this object to the SQLBaseCommand object before execution.

Properties

DbType: DbType

Get/Set. The CLR datatype of the parameter.

String: ParameterName

Get/Set. The name that the command object uses to refer to the parameter.

String: SourceColumn

Get/Set. Links this parameter to a column in a DataSet object.

Object: Value

Get/Set. The actual value assigned to this parameter.

SQLBaseParameters object

This object is basically a collection of SQLBaseParameter objects. The Add method is overloaded in five ways.

Add

Syntax **SQLBaseParameter Add (value)**

Description Add

Parameters value object.

Syntax **SQLBaseParameter Add (value)**

Description text

Parameters value SQLBaseParameter

Syntax **SQLBaseParameter Add (parametername, type)**

Description text

Parameters parametername string. Name of the parameter
type DbType. Datatype of the parameter.

Syntax **SQLBaseParameter Add (parametername, value)**

Description text

Parameters parametername string. Name of the parameter.
value object. Value of the parameter.

Syntax **SQLBaseParameter Add(parameterName, dbType, column)**

Description text

Parameters parametername string. Name of the parameter.
type DbType. Datatype of the parameter.
column string. Data set column associated with the parameter.

SQLBaseException object

This object handles all errors returned by SQLBase.

SQLBaseException Properties

int: ErrorCode

Get only. The SQLBase numeric code of the error.

int: ErrorPosition

Get only. The offset in the command text where the error was detected.

String: ErrorMessage

Get only. The description of the error.

SQLBase classes and transaction state

It is possible in the .NET framework to write code that could disrupt the state of a SQLBase transaction. To prevent this disruption, the SQLBaseCommand class will throw an exception if you try to change its Transaction property to anything other than the result of the last call to SQLBaseConnection.BeginTransaction:

```
"Failed to set transaction for the command. The transaction is not compatible with the connection."
```

The exception guards against situations like the example below:

- Open a SQLBaseConnection object named "conn"
- Create a SQLBaseCommand object against conn named "cmd"
- Create a SQLBaseTransaction object named "txn" by calling the conn.BeginTransaction method. (Now this database connection is logically "inside" a transaction.) The commands belonging to the connection, including "cmd", will acquire "txn" in their Transaction property.
- Set the cmd.Transaction property to null. (This would mean that the command no longer participates in the "txn" transaction.)
- Execute the command.
- Commit – but how? The "Commit" method is part of the SQLBaseTransaction class, and we have set cmd.Transaction to null, so we have no object to use for calling that method.

Because you changed the SQLBaseCommand.Transaction property, the SQLBase connection is now in an inconsistent state, running a pending SQLBase engine

transaction that can't or won't be committed because the necessary .NET transaction is no longer associated with that connection.

The .NET interface definitions imply that you can have a command object whose `Transaction` property points to a transaction created under some other database connection, not the connection used when the command was initialized. Despite this theoretical possibility, Gupta recommends that you *think of the connection, transaction and command objects as a set that reference each other*. This is a more accurate reflection of how the SQLBase engine works internally – a commit on a SQLBase connection commits all the cursors that are active against that connection. So, from the .NET viewpoint, a call to `cmd.Transaction.Commit` will commit not just “cmd”, but all other commands that belong to the same connection.

Note that this concept also applies if your application is running some commands under a transaction and other commands without a transaction. In this situation, the commands belonging to the transaction need a separate connection of their own.

To summarize this issue in two brief rules:

1. Whatever its current value, you must not change the `SQLBaseCommand.Transaction` property to a different value. If you attempt to do this, an exception will be thrown.
2. If you wish to have multiple transactions active simultaneously in your application, make sure that you have a separate `SQLBaseConnection` object associated with each transaction, and make sure that the `SQLBaseCommand` objects relevant to that transaction also belong to the same connection. A transaction and a connection have a one-to-one relationship that must not be altered.

Chapter 9

Introduction to the JDBC driver

This chapter describes specific issues relating to the Gupta SQLBase JDBC (Java Database Connectivity) Driver. For general information on JDBC Drivers, we recommend you visit www.javasoft.com or read one of the many published books on JDBC Drivers.

Introduction

The SQLBase JDBC driver supports applications written using JDBC 1.3.1 and higher. It is a Type 4 Driver according to the definition published by Javasoft, meaning that it is a native-protocol driver which converts JDBC calls into the network TCP/IP protocol used by the database server, requiring no additional middle layer components of either the client or the server. The SQLBase JDBC Driver is therefore both thin and portable.

Using this driver, developers can create Java applets that connect to remote SQLBase databases resident on Web Servers, or create Java applications that connect to SQLBase databases on either local or remote machines, using SQLBase's existing TCP/IP protocol support.

JDK Versions

There are multiple versions of the Java Development Kit (JDK). This driver is compatible with JDK 1.1.x and higher.

Modern Web browsers such as Netscape Communicator version 4 and higher, and Microsoft Internet Explorer version 4 and higher, support JDK 1.1.x, including JDBC functionality.

Installation

Tested Platforms

This driver has been tested on Windows 98, ME, NT, 2000, Server 2003, and XP.

NetWare server users should install Intranetware to allow Java applets and applications to communicate with the NetWare SQLBase Server.

Getting Started

You can request JDBC support during the initial install of SQLBase. You can also request it in a later install run using the “Modify” choice from the InstallShield wizard. The default directory for SQLBase itself is `c:\program files\gupta`. Presuming you also install the JDBC driver to this directory, then after installation, the following files and directories will be found.

`c:\program files\gupta\Java\SQLBaseJDBC.jar` - Contains the JDBC class files.

`c:\program files\gupta\Samples` - Contains a set of sample JDBC application programs as tutorials.

CLASSPATH

To allow the Java Virtual Machine to load the SQLBase JDBC Driver locally or at the host-of-origin, the standard Java system environment variable CLASSPATH must be modified to include the root directory containing the JDBC classes. For example, if you chose the SQLBase default directory as your installation directory, you would set the variable as follows:

```
SET CLASSPATH=%CLASSPATH%;C:\program files\gupta\Java\SQLBaseJDBC.jar
```

Server Configuration

The server should be configured to listen on TCP/IP protocols. For more information regarding the use of TCP/IP, please refer to the SQLBase Database Administrator's Guide chapter on communications.

SQLBase Support Issues

SQL Operations Support

The SQLBase JDBC Driver is designed to access relational database over the Internet or an Intranet. It currently supports common DDL and DML operations.

DDL Operations Support

All DDL operations, such as creating tables, dropping tables, and creating indices are supported. Operations which require a server connection, such as loading and unloading, backup and restore are not supported in this release of the JDBC Driver since such connections are not supported by JDBC.

DML Operations Support

Most DML operations such as insertions, deletions, updates, selects (including data binding), long varchars, and stored procedures are supported. The "ADJUSTING" clause of the INSERT keyword is not supported.

Multiple Connection Support

The SQLBase JDBC Driver supports the multiple connection model of SQLBase 7.0 and higher. An application can establish multiple connections ("cursor handles") to a SQLBase Server, and transactions committed for one connection are completely independent of transactions committed for another connection.

Multi-threading Support

The JDBC Driver is implemented to be thread safe. An application can use multiple threads to perform operations on various tables while the user continues to interact with the application.

It is NOT recommended that JDBC objects (i.e. Statement, ResultSet etc.) are shared between threads because an object's internal state can be corrupted by interleaved threads. Each thread should operate privately on JDBC objects.

Transaction Model

The transaction model is uniform between the SQLBase ODBC Driver and the JDBC Driver.

Isolation Level	Description
TRANSACTION_NONE	AutoCommit plus RL (release locks) isolation

Isolation Level	Description
TRANSACTION_READ_COMMITTED	CS (cursor stability)
TRANSACTION_SERIALIZABLE	Not Supported (Creates an Exception)
TRANSACTION_READ_UNCOMMITTED	Not Supported (Creates an Exception)
TRANSACTION_REPEATABLE_READ	RR

For SQLBase specific applications or applets that require a strict SQLBase transaction model, the SQLBase JDBC Driver provides an extended TRANSACTION MODE as indicated below:

Isolation Level	Description
TRANSACTION_SQLBase_RL	RL (release locks) isolation
TRANSACTION_SQLBase_RR	RR (read repeatability) isolation
TRANSACTION_SQLBase_CS	CS (cursor stability) isolation
TRANSACTION_SQLBase_RO	RO (read only) isolation

The default TRANSACTION isolation for the JDBC Driver is TRANSACTION_REPEATABLE_READ.

URL formats

Examples of use of the standard formats are illustrated below:

```
jdbc:sqlbase://localhost[:2157]/island [;ims=;oms=]
jdbc:sqlbase://cchan1[:2157]/island [;ims=;oms=]
jdbc:sqlbase://198.95.xxx.xx:2156/island [;ims=;oms=]
```

URL keywords:

jdbc:sqlbase: is the JavaSoft registered URL keyword for SQLBase

localhost is used to refer to the host name of the local machine

cchan1 is the host name of the machine cchan1

The default port used by SQLBase to receive incoming TCP/IP requests is **2155**. It needs to be specified only when a different port number is being used by the SQLBase server, in which case it follows the host name or IP address, separated by a colon.

island is the database where this URL is intended to connect.

198.95.xxx.xx is an IP address.

ims (optional) is input message buffer size. Use a value from 1 to 32000. If no value or an invalid value is specified, the default of **ims=2000** will be used.

oms (optional) is output message buffer size. Use a value from 1 to 32000. If no value or an invalid value is specified, the default of **oms=1000** will be used.

Data types and conversions

SQLBase data types do not map completely to Java data types. The following conversion table is provided to show the mapping between SQLBase storage types and the Java data types:

Explanation of keys:

Key	Description
C	Conversion is done - a conversion warning is issued for accuracy check
D	Date crash is done
E	Throw exception - this conversion is not allowed
W	Field width and data width mismatch - a truncation warning is issued for loss of digits
*	No warning - a valid get/set is performed

SQLBase Supported Data Types:

JDBC Data Type	SQL Data Type
CHAR	CHAR
NUM	NUMBER, INTEGER, DECIMAL, FLOAT, DOUBLE, REAL
DATE	DATE
LONG	LONGVARCHAR
DT	DATETIME
TIME	TIME

For READING Operations using ResultSet:

CHAR	NUM	DATE	LONG	DT	TIME	ResultSet Methods
W	W	W	W	W	W	getString from CHAR,VARCHAR
C	*	E	E	E	E	getBigDecimal from NUMERIC
C	C	E	E	E	E	getBool from BIT
C	W	E	E	E	E	getByte from TINYINT
C	W	E	E	E	E	getShort from SMALLINT
C	W	E	E	E	E	getInt from INTEGER
C	W	E	E	E	E	getLong from BIGINT
C	W	E	E	E	E	getFloat from REAL
C	W	E	E	E	E	getDouble from DOUBLE
*	C	C	*	C	C	getBytes[] **
C	C	*	E	C	C	getDate from DATE
C	C	C	E	C	*	getTime from TIME
C	C	C	E	*	C	getTimestamp from TIMESTAMP
*	C	C	*	C	C	getStream from LONGVARCHAR

** same as getStream from VARBINARY or LONGVARBINARY

Bind variable support

The SQLBase JDBC Driver supports the bind variable escape character '?' in a SQL statement. The driver attempts to parse the '?' from the input SQL statement. The parser skips over '?' characters that are used within single quoted expression constants or double quoted identifier to avoid disruption of the syntax of the SQL statement.

Stored procedure support

The SQLBase JDBC Driver supports the CallableStatement { call ?,... } syntax. However, since SQLBase Stored Procedures do not support any OUT parameters, a SQLException will be thrown when the CallableStatement.registerOutParameter(...) method is called. This restriction also invalidates the [? =] portion of the { [? =] call ?,... } syntax.

SQLBase stored procedures return the result via a ResultSet. After executing the CallableStatement, a getResult() method can be called to retrieve result data.

ResultSetMetaData is also helpful in determining the column types, column lengths and other information about the result data.

To provide input parameters, the CallableStatement.setXXX() methods can be called to pass Java data types from the caller to the stored procedure as bind data.

Named Cursor Support

Cursor naming is supported by the SQLBase JDBC Driver. Cursor names are generally used in conjunction with the WHERE CURRENT OF clause of an UPDATE or DELETE statement. After the cursor name is set for a Statement or PreparedStatement object, the executeQuery() can be called to prepare a SELECT statement.

The cursor name can then be used by other Statement or PreparedStatement objects with the WHERE CURRENT OF clause in SQL statements.

Error Message Support

This release provides a simplified version of error message text from error.sql. However, in order to keep the memory consumption down and the SQLBase JDBC thin, error tokens, reason and remedy are not supported.

Application and Applet Development

Java Virtual Machine reserves the `java.xxx.xxx` package for its own libraries. Generally, any library that is not part of the JVM cannot use the `java.xxx.xxx` package. The `java.xxx.xxx` package is critical to applets. Loading a library from unrecognized `java.xxx` will result in "class not found" exception being thrown by the `SecurityManager`. To avoid this problem, the `SQLBase JDBC Driver` is normally packaged under the `jdbc.gupta.sqlbase.*` package.

If you encounter problems during execution of the sample Java programs, you should examine `CLASSPATH` to ensure that it contains a path to the location of the `SQLBase JDBC Driver` classes.

To allow the `SQLBase` error text to be displayed by the `JDK 1.1.x` line of the `SQLBase JDBC Driver`, the `error.html` file must be copied to the `WWW` root directory or the `<HOME>` directory of the system where the `SQLBase JDBC Driver` is installed, in this case the host-of-origin.

Restrictions

All of the documented `JDBC API` is included in this release.

Certain functionality of the `API` is neither completely supported, directly relevant nor necessarily supportable by `SQLBase`. Where this occurs, `SQLException` will be thrown with the "...not supported..." message.

Network Check

Network check (netcheck) is not implemented in the `SQLBase JDBC Driver`. If network check is set in the `SQL.INI` file for the `SQLBase` server, the `JDBC driver` may not be able to communicate with the server.

SQLWarning Support

In general, `SQLBase` does not support warnings. However, the warning processing using `SQLWarning` is implemented in the `SQLBase JDBC Driver` for trapping potential problems generated during data conversion.

Pre 6.0 SQLBase Support

`SQLBase JDBC Driver` communicates with `SQLBase` via the latest version of its network protocol in order to support stored procedures. As a result, `SQLBase JDBC Driver` does not support any version of `SQLBase` prior to release 6.0.

NLS Databases

The `JDBC Driver` supports non-US databases. Please refer to the file `CountrySetting.java` in the `samples` directory for more information on how to

connect to non-US databases. DBCS databases, however, are not supported in this release.

Glossary

A

Abort To return the values changed by a transaction to their original state.

Accessor A collection of information that describes how data is stored in the consumer's buffer. The provider uses this information to determine how to transfer data to and from this buffer. See also reference accessor.

Accessor handle A handle that identifies an accessor.

Auto-commit mode A transaction commit mode in which all actions taken in a transaction are committed immediately after they are performed.

B

Binary large object (BLOB) Any binary or character data over a certain number of bytes, such as 255. Typically much longer. Such data is generally sent to and retrieved from the data source in parts. Also known as long data.

Binding As a verb, the act of associating a column in a rowset or a parameter in a text command with a consumer variable. As a noun, the association or the DBBINDING structure that describes the association.

Bookmark A value that identifies a row in a rowset. Bookmarks are saved by the consumer and used later in the life of the rowset to retrieve a particular row.

BSTR A pointer to a null-terminated character string in which the string length is stored with the string.

B-tree A tree structure for storing database indexes.

Buffer A piece of memory used to pass information, usually data, between the consumer and provider.

C

Catalog A database object that contains one or more schemas.

Change To update, delete, or insert a row of data.

Class ID A globally unique identifier (GUID) associated with an OLE class object. If a class object will be used to create more than one instance of an object, the associated server application should register its CLSID in the system registry so that clients can locate and load the executable code associated with the object(s). Every OLE server or container that allows linking to its embedded objects must register a CLSID for each supported object definition.

Client/server A database access strategy in which one or more clients access data through a server. The clients generally implement the user interface while the server controls database access.

Column The container for a single item of information in a row. Also known as field.

Column ID A structure used to identify a column, primarily in a command where there are no stable ordinals or column names.

COM object An object that conforms to the OLE Component Object Model (COM). A COM object is an instance of an object definition, which specifies the object's data and one or more implementations of interfaces on the object. Clients interact with a COM object only through its interfaces. See also Component Object Model and interface.

Command An OLE DB object that encapsulates a command.

Command text The text command associated with a command object. Although it is not required to be, this is usually an SQL statement.

Commit To make the changes in a transaction permanent.

Component An object that encapsulates both data and code, and provides a well-specified set of publicly available services.

Component Object Model (COM) The OLE object-oriented programming model that defines how objects interact within a single process or between processes. In COM, clients have access to an object through interfaces implemented on the object. See also interface.

Concurrency The ability of more than one transaction to access the same data at the same time.

Consumer Software that calls OLE DB methods and interfaces.

Container A file containing linked or embedded objects.

Convert To change data from one type to another, such as from an integer to a string.

Coordinated transaction A transaction composed of multiple, subordinate transactions. All of the subordinate transactions must succeed for the coordinated transaction to succeed. Also known as a distributed transaction.

CoType A way to define a group of COM objects, such as rowsets or commands, that have similar characteristics. All COM objects that belong to a particular CoType must expose the mandatory interfaces in that CoType. In addition, they can expose the optional interfaces in the CoType and any interfaces not in the CoType.

D

Data The data for a parameter in a text command or a column in a row.

Data Definition Language (DDL) Those text commands that define, as opposed to manipulate, data. For example, the SQL statements CREATE TABLE, CREATE INDEX, GRANT, and REVOKE.

Data Manipulation Language (DML) Those text commands that manipulate, as opposed to define, data. For example, the SQL statements INSERT, UPDATE, DELETE, and SELECT.

Data part Data has three parts: the data value, the length of the data value, and the status of the data value.

Data provider A provider that directly exposes data, as opposed to a service component.

Data source The data the user wants to access, such as the data in a database, file, or array. This is distinct from a data source object.

Data source object An OLE DB object that connects to a data source. This is distinct from the data source.

Data type The type of a piece of data. See also type indicator.

Database A discrete collection of data in a DBMS. Also a DBMS.

Database Management System (DBMS) A layer of software between the physical database and the user. The DBMS manages all access to the database. An OLE DB provider can be built directly on top of a DBMS or as a layer between the DBMS and the consumer.

Deferred column A column for which the data is not instantiated in the rowset until the consumer actually attempts to access that data.

Delayed update mode An update mode in which changes made through IRowsetChange are cached in the rowset and transmitted to the data source only when IRowsetUpdate::Update is called. See also immediate update mode.

Delete To remove an existing row of data from the data source.

Distributed transaction See coordinated transaction.

Dynamic error An error message that cannot be predicted because it is not provider specific.
Dynamic error ID An identifier that is used by a lookup service and is associated with an error object.

E

Empty string A zero-length string. See also null pointer and NULL value.

Endogenous change A change to data originating from a process other than your own.

Enumerator An OLE DB object that searches for data sources and other enumerators. See also root enumerator.

Error A condition in which a fatal error occurred. Also used to refer to any error, regardless of whether it is fatal.

Error code A class of return code, of the type HRESULT, that begins with E_ or DB_E_ and indicates that the method failed completely and was unable to do any useful work.

Error lookup service A method used by a provider to handle and interpret OLE DB error objects.

Error object An object that contains detailed information about an error. See also OLE Automation error object and OLE DB error object.

Event An action taken by a provider, such as changing a column value or deleting a row, of which the provider notifies the consumer.

Execution plan A plan generated by a query engine to execute a command. Equivalent to executable code compiled from a third-generation language such as C.

F

Fetch To retrieve one or more rows from the data source and instantiate them in a rowset.

Field See column.

Fixed-length data type A data type that is always stored in the same number of bytes, such as a two-byte integer. See also variable-length data type.

Foreign key A column or columns in a table that match the primary key in another table.

G

Getting data Transferring data from the provider to the consumer, as in getting column or output parameter data.

Globally Unique Identifier (GUID) A 16-byte value that uniquely identifies something, usually the software that implements one or more COM objects or an interface on one of those objects. Also known as a UUID (Universally Unique Identifier).

H

Handle A value that uniquely identifies something such as a row or an accessor. Handles are meaningful only to the software that creates and uses them, but are passed by other software to identify things.

HRESULT An opaque result handle defined to be zero for a successful return from a function and nonzero if error or status information is returned.

I

Interface Identifier (IID) A globally unique identifier (GUID) associated with an interface. Some functions take IIDs as parameters to allow the caller to specify which interface pointer should be returned.

Immediate update mode An update mode in which changes made through `IRowsetChange` are immediately transmitted to the data source. See also delayed update mode.

Index rowset A rowset built over an index. Each row in an index rowset contains a bookmark that points to a row in a rowset built over the corresponding table.

Initialize To change the state of an enumerator or data source object so it can be used to access data. For example, initializing a data source object might require the provider to open a data file or connect to a database.

Input parameter A parameter in a text command for which the consumer supplies a value to the provider.

Input/output parameter A parameter in a text command for which the consumer supplies a value to the provider and the provider returns a value to the consumer.

Insert To add a new row of data to the data source.

Instantiate To create an instance of a COM object.

Interface A group of semantically related functions that provide access to a COM object. Each OLE interface defines a contract that allows objects to interact according to the Component Object Model (COM). Although OLE provides many interface implementations, most interfaces can also be implemented by developers designing OLE applications. See also Component Object Model and COM object.

IPersist* object An OLE object that supports IPersistStream, IPersistStreamInit, or IPersistStorage.

Isolation level See transaction isolation level.

J

JDBC A set of Java classes which standardizes access to any vendor's database from Java applications or applets.

Join An operation in a relational database that links the rows in two or more tables by matching values in specified columns.

K

Key A column or columns whose values identify a row. See also primary key and foreign key.

Key value bookmark A bookmark that uses a unique key to identify a row. See also numeric bookmark.

L

Length The length of a data value. See also data part, value, and status.

Listener A consumer that has requested that a provider send it notifications of various events.

Long data See BLOB.

M

Manual-commit mode A transaction commit mode in which transactions must be explicitly committed or aborted by calling ITransaction::Commit or ITransaction::Abort.

Maximum precision The maximum number of base 10 digits that can be stored in a particular data type. See also precision.

Metadata Data that describes a parameter in a text command or a column in a rowset. For example, the data type, length, and updatability of a column.

Method A function in an interface.

Moniker A name for a specific object instance.

Multiple results object An OLE DB object created by executing a command and through which multiple results (rowsets or row counts) can be retrieved.

N

Next fetch position The position of the next row that will be fetched by a call to IRowset::GetNextRows.

Notification A call from a provider to a consumer, in which the provider notifies the consumer that a particular event is occurring.

Notification sink An actual object that implements the notification interface.

Null pointer A pointer with a value of zero. It is an error to dereference a null pointer. See also empty string and NULL value.

NULL value Having no explicitly assigned value. In particular, a NULL value is different from a zero or a blank. See also empty string and null pointer.

Numeric bookmark A bookmark that uses a unique number to identify a row. See also key value bookmark.

O

Object In OLE, a programming structure encapsulating both data and functionality that are defined and allocated as a single unit and for which the only public access is through the programming structure's interfaces. A COM object must support, at a minimum, the IUnknown interface, which maintains the object's existence while it is being used and provides access to the object's other interfaces. See also Component Object Model and interface.

ODBC Microsoft's earliest standard for

OLE Microsoft's object-based technology for sharing information and services across process and machine boundaries.

OLE Automation error object An error object that conforms to the standards specified for such objects by OLE Automation. See also OLE DB error object.

OLE DB A set of interfaces that expose data from a variety of data sources using COM.

OLE DB error object An error object used by OLE DB objects to return an error. OLE DB error objects are an extension of Automation error objects.

OLE DB object A COM object defined by OLE DB. The COM objects defined by OLE DB are enumerators, data source objects, sessions, commands, rowsets, multiple results objects, OLE DB error objects, transaction objects, and transaction options objects.

OLE DB Software Development Kit (SDK) A collection of redistributable software, header files, tools, sample code, and documentation to be used by developers of OLE DB consumers and providers.

OLE object In OLE DB, a COM object that is stored in a column. In OLE, a COM object.

Optimistic concurrency A strategy to increase concurrency in which rows are not locked. Instead, before they are updated or deleted, a rowset checks to see if they have been changed since they were last read. If so, the update or delete fails. See also pessimistic concurrency.

Ordinary bookmark A bookmark whose value is defined by the provider. See also standard bookmark.

Output parameter A parameter in a text command for which the provider returns a value to the consumer.

P

Parallel processing A method of processing that can run only on a computer that contains two or more processors running simultaneously.

Parameter A variable in a text command. A parameter can be an input, input/output, or output parameter.

Pending change A change that has been cached in a rowset and not yet transmitted to the data source. See also delayed update mode and immediate update mode.

Persist To save the current state of a COM object, such as to a file. Currently, only data source objects can be persisted.

Pessimistic concurrency A strategy for implementing serializability in which rows are locked so that other transactions cannot change them. See also optimistic concurrency.

Phase A step in a sequence of notifications caused by a single event. The sequence of notifications is similar to the phases in a two-phase commit protocol.

Precision The number of base 10 digits in a number. See also maximum precision.

Prepare To compile a command. An execution plan is created by preparing a command.

Preserved rowset A rowset in which all previously defined functionality is retained after a commit or abort.

Primary key A column or columns that uniquely identifies a row in a table.

Procedure A group of one or more precompiled commands (generally SQL statements) that are stored as a named object in a database.

Property Attributes of an OLE DB object. For example, the maximum number of rows in a rowset that can be active at one time.

Property group The set of all properties that apply to a particular OLE DB object.

Property set A property is identified by a GUID and an integer (the property ID). A property set is the set of all properties that share the same GUID.

Provider Software that implements OLE DB methods and interfaces.

Q

Query A text command. Sometimes used to mean a text command that creates a rowset.

R

Reason The specific event that occurred, such as changing a row value or deleting a row.

Record See row.

Reference counting Keeping a count of each interface pointer held on an object to ensure that the object is not destroyed before all references to it are released. In OLE DB, rows and accessors are also reference counted.

Reference accessor An accessor that enables a consumer to get rowset data directly from the provider's buffer. Support for reference accessors is optional.

Release To decrease the reference count on a row, accessor, or COM object. When the reference count reaches zero, the provider generally releases the resources used by the row, accessor, or COM object.

Result A row count or rowset created by executing a command. See also multiple results object.

Resynchronize To update the data in a rowset with the data in the data source that is visible to the current transaction according to its isolation level.

Return code The value returned by an OLE DB method.

Root enumerator An enumerator shipped in the OLE DB SDK that enumerates the data sources and enumerators listed in the registry. See also enumerator.

Row A set of related columns that describe a specific entity. Also known as a record.

Row handle A handle used to identify a row.

Rowset An OLE DB object that contains a set of rows, each of which has columns of data.

S

Scale The number of digits in a number that are to the right of the decimal point.

Schema A database object that contains one or more tables, often created by a single user.

Schema rowset A predefined rowset that provides information about the structure of a database.

Self bookmark A bookmark, stored in column 0 of a row, that is used to return to that row.

Serializability Whether two transactions executing simultaneously produce a result that is the same as the serial (or sequential) execution of those transactions. Serializable transactions are required to maintain database integrity.

Service component A provider that does not directly expose data, but instead provides a service, such as query processing. Used in conjunction with data providers.

Session An OLE DB object that serves as the context for a transaction.

Setting data Transferring data from the consumer to the provider, as in setting column or input parameter data.

SQL Structured Query Language. A language used by relational databases to query, update, and manage data. Text commands often use SQL.

Standard bookmark A bookmark whose value is defined by OLE DB. See also ordinary bookmark.

Static error An error (text) that is stored by a specific provider lookup service, and most commonly tied to a single data store.

Status The status of a data value. See also data part, value, and length.

Storage interface An interface used to access data in a storage object: ISequentialStream, IStream, IStorage, or ILockBytes.

Storage object A COM object that implements a storage interface. Storage objects are used to access BLOB data and OLE objects stored in a column.

Success A condition in which no errors occurred.

Success code A class of return code, of the type HRESULT, that begins with S_ or DB_S_ and indicates success of the method.

T

Table A collection of rows in the data source.

Text command A text string, usually an SQL statement, that defines a command.

Transaction An atomic unit of work. The work in a transaction must be completed as a whole; if any part of the transaction fails, the entire transaction fails.

Transaction isolation The act of isolating one transaction from the effects of all other transactions.

Transaction isolation level A measure of how well a transaction is isolated.

Transaction object An OLE DB object used to support transactions.

Transaction options object An OLE DB object used to define various options for a transaction.

Transfer To move data between the consumer's and provider's buffers. The provider, not the consumer, transfers data. See also getting data and setting data.

Transmit To send changes made through IRowsetChange to the data source. See also delayed update mode and immediate update mode.

Transmitted change A change that has been sent to the data source. See also pending change.

Truncate To discard one or more bytes of variable-length data or non-significant digits of numeric data. Truncation results in a warning condition when getting data and an error condition when setting data.

Type indicator An integer value passed to or returned from an OLE DB method to indicate the data type of a consumer variable, a parameter, or a column.

U

Uninitialize To change the state of an enumerator or data source object so it cannot be used to access data. For example, uninitializing a data source object might require the provider to close a data file or disconnect from a database.

Unprepare To discard the current execution plan.

Update To change an existing row of data in the data source. Also, to transmit pending changes to the data source.

User The end user, which is a generally a person, as opposed to the consumer, which is a piece of software.

V

Value A data value. See also data part, length, and status.

Variable-length data type A data type for which the length of the data can vary, such as a string. See also fixed-length data type.

Visibility Whether data values can be detected by a rowset. Refers both to the visibility of data in a data source and data cached in a rowset.

W

Warning A condition in which a nonfatal error occurred.

Warning code A class of return code, of the type HRESULT, that begins with S_ or DB_S_ and indicates success of the method but with a warning.

Z

Zombie A state in which the only valid consumer action on a COM object is generally to release that object.

.NET

- exceptions 8-14
- transactions 8-14

A

- ActiveX Data Objects 4-3, 5-1
- ADO 4-3, 5-2
 - coding sample 5-7
- adoid.h 5-3
- adoint.h 5-3
- adojavas.inc 5-3, 5-5
- adovbs.inc 5-3
- Advanced Data Connector 5-2
- apartment-model 5-3
- applet 9-8
- application developers 1-10

B

- batch updates 5-2, 5-5
- BOF 5-9

C

- C++ 5-3
- caching 5-2
- CLASSPATH 9-2
- closing 5-10
- collections 5-4, 7-2
- COM 4-2
 - cotypes 6-2
 - objects 6-1
- COM+ 1-9
- command
 - COM object 6-2
 - interfaces 6-6
- command object 5-3, 7-3
- CommandText 8-6
- CommandText property 8-6
- CommandType property 8-6
- Common Language Runtime 8-2
- Component Object Model 4-2
- configuring the provider 4-4
- connecting 5-6
 - using ADO 5-7
- Connection 8-5
- connection lifetime 8-3
- connection object 5-3, 7-3
- Connection pooling 8-5
- connection string 8-5

- Connectivity Administrator 1-2
- Crystal Reports 4-4
- cursor object 5-4
- cursors 5-2

D

- Data Access Objects 5-2
- data consumer 4-2
- data provider 4-2
- data source 4-4, 5-9
 - COM object 6-2
 - interfaces 6-3
- Datatypes 8-2
- DBExplore.exe 5-6
- DBPROP_INIT_PROVIDERSTRING 6-3
- DeleteCommand property 8-11
- Delphi 4-4
- disconnect 5-10
- Dynamic Properties 5-4

E

- EOF 5-9
- error
 - COM object 6-2
 - interfaces 6-9
 - routine 5-10
 - trapping 5-9
- error object 5-4, 7-4
- Errors collection 5-7, 5-8
- escape character 9-7

F

- fetch 5-9
- field object 5-4, 7-5

H

- header file 5-5
- hierarchy 5-3

I

- IEnumerable 8-8
- IEnumerator 8-8
- InsertCommand property 8-11
- ISLAND sample database 5-5, 5-6
- IsNull 5-8

J

- Java 5-3

- Java Development Kit 9-1
- Java Type Library Wizard 5-3
- Java Virtual Machine 9-8
- JDBC 9-1
 - bind variables 9-7
 - data types 9-5
 - isolation levels 9-3
- JScript 5-3, 5-5

L

- last record 5-11
- libraries
 - ADO 5-3
- Long Binary 5-6
- Long text fields 5-6

M

- methods 5-4, 7-2
- Microsoft Visual J++ 5-3
- MTS 1-9

N

- named cursor 9-7
- navigation 5-10
- NULL values 5-9

O

- objects
 - ADO 5-3
- ODB
 - data types 2-6
- ODBC 2-1
 - see Chapter 10
 - Conformance 2-3
 - connection string 3-1
 - data source properties 2-7
 - driver 2-2
 - driver manager 2-2
 - isolation 2-4
 - overview 2-2
 - supported functions 2-4
- ODBC API 4-2
- Open DataBase Connectivity
 - see ODBC

P

- parameter object 5-4, 7-6
- password 4-4, 5-9

- poolsize 8-3
- PowerBuilder 4-4
- previous record 5-11
- program ID 5-3
- properties 5-4, 7-2
- Properties collections 5-6
- property object 5-4
- provider name 4-4, 5-5
- provider string 5-9

R

- reading 4-2
- Recordset object 5-5
- recordset object 5-4, 7-7
- Remote Data Objects 5-2
- Remote Data Service 5-2
- RowPos 8-8
- rowset
 - COM object 6-2
 - interfaces 6-8

S

- sample applications 5-6
- schema 5-6
- Scripting code 5-5
- scrollable 5-5
- scrolling result sets 8-8
- SelectCommand property 8-11
- session
 - COM object 6-2
 - interfaces 6-5
- simplefetch 5-6, 5-7
- SQL.INI 1-2
- SQL/API functions 4-2
- SQLBaseCommand
 - Cancel 8-6
 - GetSQLBaseParameter 8-7
 - SetSQLBaseParameter 8-7
- SQLBaseCommand object 8-6
- SQLBaseConnection 8-3
 - BeginTransaction 8-4
 - ChangeDatabase 8-4
 - Close 8-4
 - CreateCommand 8-4
 - GetSQLBaseSchemaTable 8-4
 - Open 8-5
- SQLBaseDataAdapter object 8-10
- SQLBaseDataReader

- Get methods 8-8
- Get methods for long (BLOB) data 8-8
- GetDataTypeName 8-9
- GetFieldType 8-9
- GetName 8-9
- GetOrdinal 8-9
- GetValues 8-10
- IsDBNull 8-10
- NextResult 8-10
- SQLBaseDataReader object 8-8
- SQLBaseParameter object 8-12
- SQLBaseParameters
 - Add 8-13
- SQLBaseParameters object 8-13
- SQLBaseSchemaGuid 8-5
- SQLBaseTransaction
 - Commit 8-12
 - Rollback 8-12
- SQLBaseTransaction object 8-11
- stored procedure 9-7
- stored procedures 5-2
- Supported ODBC Functions 2-4

T

- TCP/IP 1-9
- TextBox objects 5-8
- Threading 5-3
- Transaction Model 9-3
- transaction state 8-14

U

- Universal Data Access 4-2
- update 5-10
- UpdateCommand property 8-11

V

- VBScript 5-3
- Visual Basic 4-4

W

- Web sites 5-1

