



Users Guide

**Adaptive Server® Enterprise
ODBC Driver by Sybase**

15.0

[Microsoft Windows, Linux, and Mac OS X]

DOCUMENT ID: DC20116-01-1500-03

LAST REVISED: November 2008

Copyright © 2008 by Sybase, Inc. All rights reserved.

This publication pertains to Sybase software and to any subsequent release until otherwise indicated in new editions or technical notes. Information in this document is subject to change without notice. The software described herein is furnished under a license agreement, and it may be used or copied only in accordance with the terms of that agreement.

To order additional documents, U.S. and Canadian customers should call Customer Fulfillment at (800) 685-8225, fax (617) 229-9845.

Customers in other countries with a U.S. license agreement may contact Customer Fulfillment via the above fax number. All other international customers should contact their Sybase subsidiary or local distributor. Upgrades are provided only at regularly scheduled software release dates. No part of this publication may be reproduced, transmitted, or translated in any form or by any means, electronic, mechanical, manual, optical, or otherwise, without the prior written permission of Sybase, Inc.

Sybase trademarks can be viewed at the Sybase trademarks page at <http://www.sybase.com/detail?id=1011207>. Sybase and the marks listed are trademarks of Sybase, Inc. ® indicates registration in the United States of America.

Java and all Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

Unicode and the Unicode Logo are registered trademarks of Unicode, Inc.

All other company and product names used herein may be trademarks or registered trademarks of their respective companies.

Use, duplication, or disclosure by the government is subject to the restrictions set forth in subparagraph (c)(1)(ii) of DFARS 52.227-7013 for the DOD and as set forth in FAR 52.227-19(a)-(d) for civilian agencies.

Sybase, Inc., One Sybase Drive, Dublin, CA 94568.

Contents

About This Book	vii
-----------------------	-----

CHAPTER 1	Introduction to ODBC Programming	1
	Introduction to ODBC	2
	ODBC conformance	2
	ODBC Driver Manager	3
	Using the ASE ODBC Driver samples.....	6
	Defining ODBC handles	6
	Allocating ODBC handles	8
	Connecting to a data source	9
	Choosing an ODBC connection function	9
	Establishing a connection.....	10
	Using threads and connections in ODBC applications.....	12
	Executing SQL statements.....	12
	Executing statements directly.....	13
	Executing statements with bound parameters	13
	Executing prepared statements.....	15
	Working with result sets	17
	Choosing cursor characteristics	17
	Retrieving data	18
	Updating and deleting rows through a cursor.....	18
	Using scrollable cursors	19
	Calling stored procedures	23
	Handling errors.....	25
	Datatype mappings	26
	Using computed columns.....	28
	Using server-specified packet size.....	28
	Using Large Identifiers for database objects	29

CHAPTER 2	Connecting to a Database	31
	Introduction to connections	31
	Installing ODBC MetaData stored procedures	31
	How connection parameters work.....	32

Character sets	33
Configuring the ASE ODBC Driver.....	34
Windows.....	34
Linux.....	35
Mac OS X.....	37
ODBC ini files	38
Connecting using a data source.....	40
Using connection parameters.....	40

CHAPTER 3	ASE Advanced Features	45
	Asynchronous execution for ODBC	45
	Supported Adaptive Server Cluster Edition features.....	46
	Login redirection.....	47
	Connection migration	47
	Connection failover in Cluster Edition	48
	Using Distributed Transactions	49
	Programming for MS DTC.....	49
	Programming components deployed in Sybase EAServer, MTS, or COM+	50
	Connection properties for Distributed Transaction support.....	51
	Using directory services.....	52
	LDAP as a directory service	52
	Using directory services	53
	Enabling directory services	54
	Bookmark and bulk support	55
	DSN Migration tool	56
	Using the Migration tool.....	56
	Conversion switches	56
	Using password encryption	57
	Windows.....	58
	Linux.....	59
	Mac OS X.....	59
	Password expiration handling	59
	Using SSL	60
	SSL security levels in ASE ODBC Driver	62
	Validating the server by its certificate.....	62
	Enabling SSL connections	63
	Using failover in high availability systems	65
	Windows.....	67
	Linux.....	68
	Mac OS X.....	69
	Enabling Kerberos authentication	69
	Process overview	69
	Requirements	70

Enabling Kerberos authentication	71
Windows	71
Linux	72
Obtaining an initial ticket from the Key Distribution Center	72
Index	75

About This Book

Audience

This document is intended for application developers who need access to data from Adaptive Server® Enterprise (ASE) Microsoft Windows, Linux, and Mac OS X platforms, using Open Database Connectivity (ODBC).

How to use this book

The information in this book is organized as follows:

- Chapter 1, “Introduction to ODBC Programming”
- Chapter 2, “Connecting to a Database”
- Chapter 3, “ASE Advanced Features”

Related documents

For information about installing the Software Developer’s Kit (SDK), see the Software Developer’s Kit and Open Server 15.0 *Installation Guide*.

For information about known problems and recent updates, see the Software Developer’s Kit and Open Server 15.0 *Release Bulletin*.

For information about installing Adaptive Server Enterprise on Windows, see the Adaptive Server Enterprise 15.0 for Windows *Installation Guide*.

For information about installing the Adaptive Server Enterprise on Linux, see the Adaptive Server Enterprise 15.0 for Linux *Installation Guide*.

For information about known problems and recent updates to Adaptive Server Enterprise on Windows, see the Adaptive Server Enterprise for Windows *Release Bulletin*.

For information about known problems and recent updates to Adaptive Server Enterprise on Linux, see the Adaptive Server Enterprise 12.5.2 for Linux *Release Bulletin*.

Other sources of information

Use the Sybase Getting Started CD, the SyBooks™ CD, and the Sybase Product Manuals Web site to learn more about your product:

- The Getting Started CD contains release bulletins and installation guides in PDF format, and may also contain other documents or updated information not included on the SyBooks CD. It is included with your software. To read or print documents on the Getting Started CD, you need Adobe Acrobat Reader, which you can download at no charge from the Adobe Web site using a link provided on the CD.

-
- The SyBooks CD contains product manuals and is included with your software. The Eclipse-based SyBooks browser allows you to access the manuals in an easy-to-use, HTML-based format.

Some documentation may be provided in PDF format, which you can access through the PDF directory on the SyBooks CD. To read or print the PDF files, you need Adobe Acrobat Reader.

Refer to the SyBooks installation guide on the Getting Started CD, or the *README.txt* file on the SyBooks CD for instructions on installing and starting SyBooks.

- The Sybase Product Manuals Web site is an online version of the SyBooks CD that you can access using a standard Web browser. In addition to product manuals, you will find links to EBFs/Maintenance, Technical Documents, Case Management, Solved Cases, newsgroups, and the Sybase Developer Network.

To access the Sybase Product Manuals Web site, go to Product Manuals at <http://www.sybase.com/support/manuals/>.

Sybase certifications on the Web

Technical documentation at the Sybase Web site is updated frequently.

❖ Finding the latest information on product certifications

- 1 Point your Web browser to Technical Documents at <http://www.sybase.com/support/techdocs/>.
- 2 Click Certification Report.
- 3 In the Certification Report filter select a product, platform, and timeframe and then click Go.
- 4 Click a Certification Report title to display the report.

❖ Finding the latest information on component certifications

- 1 Point your Web browser to Availability and Certification Reports at <http://certification.sybase.com/>.
- 2 Either select the product family and product under Search by Base Product; or select the platform and product under Search by Platform.
- 3 Select Search to display the availability and certification report for the selection.

❖ **Creating a personalized view of the Sybase Web site (including support pages)**

Set up a MySybase profile. MySybase is a free service that allows you to create a personalized view of Sybase Web pages.

- 1 Point your Web browser to Technical Documents at <http://www.sybase.com/support/techdocs/>.
- 2 Click MySybase and create a MySybase profile.

Sybase EBFs and software maintenance

❖ **Finding the latest information on EBFs and software maintenance**

- 1 Point your Web browser to the Sybase Support Page at <http://www.sybase.com/support>.
- 2 Select EBFs/Maintenance. If prompted, enter your MySybase user name and password.
- 3 Select a product.
- 4 Specify a time frame and click Go. A list of EBF/Maintenance releases is displayed.

Padlock icons indicate that you do not have download authorization for certain EBF/Maintenance releases because you are not registered as a Technical Support Contact. If you have not registered, but have valid information provided by your Sybase representative or through your support contract, click Edit Roles to add the “Technical Support Contact” role to your MySybase profile.

- 5 Click the Info icon to display the EBF/Maintenance report, or click the product description to download the software.

Conventions

The following conventions are used in this book.

- Functions, command names, command option names, program names, program flags, properties, keywords, statements, and stored procedures are printed as follows:

You use the `SQLSetConnectAttr` function to control details of the connection. For example, the following statement turns off ODBC autocommit behavior.

- Variables, parameters, and user-supplied words are in italics in syntax and in paragraph text, are printed as follows:

For example, the following statement allocates a `SQL_HANDLE_STMT` handle the with name `stmt`, on a connection with a handle named `dbc`.

- Names of database objects such as databases, tables, columns, and datatypes, are printed as follows:

The value of the `pubs2` object.

- Examples that show the use of functions are printed as follows:

```
retcode = SQLConnect( dbc,
                      (SQLCHAR*) "MANGO", SQL_NTS,
                      (SQLCHAR*) "sa", SQL_NTS,
                      (SQLCHAR*) "", SQL_NTS );
```

Syntax formatting conventions are summarized in the following table.

Table 1: Syntax formatting conventions

Key	Definition
{ }	Curly braces mean you must choose at least one of the enclosed options. Do not include braces in the command.
[]	Brackets mean you can choose or omit enclosed options. Do not include brackets in the command.
	Vertical bars mean you can choose no more than one option (enclosed in braces or brackets).
,	Commas mean you can choose as many options as you need (enclosed in braces or brackets). Separate your choices with commas, to be typed as part of the command. Commas can also be required in other syntax contexts.
()	Parentheses are to be typed as part of the command.
...	An ellipsis (three dots) means you can repeat the last unit as many times as you need. Do not include ellipses in the command.

Accessibility features

This document is available in an HTML version that is specialized for accessibility. You can navigate the HTML with an adaptive technology such as a screen reader, or view it with a screen enlarger.

Software Developer's Kit version 15.0 and the HTML documentation have been tested for compliance with U.S. government Section 508 Accessibility requirements. Documents that comply with Section 508 generally also meet non-U.S. accessibility guidelines, such as the World Wide Web Consortium (W3C) guidelines for Web sites.

The online help for this product is also provided in HTML, which you can navigate using a screen reader.

Note You might need to configure your accessibility tool for optimal use. Some screen readers pronounce text based on its case; for example, they pronounce ALL UPPERCASE TEXT as initials, and Mixed Case Text as words. You might find it helpful to configure your tool to announce syntax conventions. Consult the documentation for your tool.

For information about how Sybase supports accessibility, see Sybase Accessibility at <http://www.sybase.com/accessibility>. The Sybase Accessibility site includes links to information on Section 508 and W3C standards.

If you need help

Each Sybase installation that has purchased a support contract has one or more designated people who are authorized to contact Sybase Technical Support. If you cannot resolve a problem using the manuals or online help, please have the designated person contact Sybase Technical Support or the Sybase subsidiary in your area.

Introduction to ODBC Programming

This chapter presents information for developing applications that call the Open Database Connectivity (ODBC) programming interface directly.

Topic	Page
Introduction to ODBC	2
Using the ASE ODBC Driver samples	6
Defining ODBC handles	6
Connecting to a data source	9
Executing SQL statements	12
Working with result sets	17
Calling stored procedures	23
Handling errors	25
Datatype mappings	26
Using computed columns	28
Using server-specified packet size	28
Using Large Identifiers for database objects	29

The primary documentation for ODBC application development is the Microsoft ODBC SDK documentation at <http://msdn.microsoft.com>. This chapter provides introductory material and describes features specific to Adaptive Server® Enterprise ODBC Driver by Sybase (referred to hereafter as ASE ODBC Driver) but is not a complete guide to ODBC application programming.

Introduction to ODBC

The Open Database Connectivity (ODBC) interface is a call-based application programming interface defined by Microsoft Corporation as a standard interface to database management systems on Windows operating systems. In addition, ODBC is now widely used on many non-Windows platforms, such as Linux.

Software requirements

To write ODBC applications for Adaptive Server Enterprise, you need:

- Adaptive Server Enterprise, versions 12.0 and above
- A C compiler capable of creating programs for your environment
- ODBC Software Development Kit
- On non-Windows platforms, there are open source projects like unixODBC and iODBC that release the required headers and libraries to build ODBC applications

Note Significant portions of this book deal with writing C programs to access data using ODBC functions with ASE ODBC Driver. There are utilities, programs, and 4GL RAD tools that can use ODBC connections. For example, you can write a PowerBuilder application or a PHP Web page that connects to an ODBC Data Source. For such uses, you only need to know how to set up a Data Source using ASE ODBC Driver. Once the Data Source has been set up, these tools completely abstract the underlying ODBC function calls.

Supported platforms

The ASE ODBC Driver is supported on:

- Windows 32-bit (x86 architecture)
- Windows 64-bit (x86-64 architecture)
- Linux 32-bit (x86 architecture)
- Linux 64-bit (x86-64 architecture)
- Mac OS X on Intel

See the Software Developer's Kit and Open Server 12.5.1 *Installation Guide*, for version details of supported platforms.

ODBC conformance

The ASE ODBC Driver conforms to ODBC 3.52 specification.

Levels of ODBC support	ODBC features are arranged according to level of conformance. Features are either Core, Level 1, or Level 2, with Level 2 being the most complete level of ODBC support. These features are listed in the <i>Microsoft ODBC Programmer's Reference</i> .
Features supported by ASE ODBC Driver	<p>The ASE ODBC Driver meets Level 2 conformance with the following exceptions:</p> <ul style="list-style-type: none">• Level 1 conformance The ASE ODBC Driver supports all Level 1 features, except for asynchronous execution of ODBC functions, SQLBulkOperations, SQLSetPos, and scrollable cursors.• Level 2 conformance The ASE ODBC Driver supports all Level 2 features, except for asynchronous execution of ODBC functions, using bookmarks.
ODBC backward compatibility	Applications developed using older versions of ODBC continue to work with the ASE ODBC Driver and the newer ODBC Driver Manager. The new ODBC features are not available for older applications.

ODBC Driver Manager

The ODBC Driver Manager manages the communications between the user applications and ODBC Drivers. Typically, user applications are linked against the ODBC Driver Manager. The Driver Manager manages the job of loading and unloading the appropriate ODBC Driver for the application. Applications make ODBC calls to the ODBC Driver Manager, which performs basic error checking and then processes these calls or passes them on to the underlying ODBC Driver.

The ODBC Driver Manager is not a required component, but it exists to solve many issues surrounding ODBC application development and deployment. Some advantages of using an ODBC Driver Manager are:

- Portable data access: Applications do not need to be rebuilt to use a different DBMS.
- Runtime binding to a data source.
- Ability to easily change a data source.

To use the ASE ODBC Driver without using the ODBC Driver Manager, you can link your application directly with the ASE ODBC Driver library. Then, the resulting executable can connect to only ASE data sources.

An ODBC Driver Manager is not included with the ASE ODBC Driver: typically, it is installed when you install the operating system. Also, multiple open source and commercial implementations of ODBC Driver Manager are available. The ASE ODBC Driver works with any ODBC Driver Manager implementation.

The ASE ODBC Driver has been tested with the following ODBC Driver Managers:

- On Windows, the Microsoft ODBC Driver Manager that is included with Windows
- On Linux, the unixODBC Driver Manager that is included with Red Hat and SuSE
- On Mac OS X, the iODBC Driver Manager that is included with Mac OS X

Building applications using an ODBC Driver Manager

You can build applications using an ODBC Driver Manager on the following operating systems:

- Windows
- Linux
- Mac OS X

Windows

The Microsoft ODBC Driver Manager includes either a DLL named *odbc32.dll* or an import library named *odbc32.lib*. On Windows 2000, the *odbc32.dll* file is located in *%SystemRoot%\system32*. The *odbc32.lib* file can appear in a number of locations, depending on which products you have installed. If you use Microsoft Visual Studio.NET, the *odbc32.lib* is located in the *%Install Path% to Microsoft Visual Studio%\Vc7\PlatformSDK\Lib*.

To link an ODBC application against the Microsoft ODBC Driver Manager, use *odbc32.lib*.

Linux

The ODBC Driver Manager includes a shared library named *libodbc.so*, which is a soft link to a library named *libodbc.so.1*. This file is typically located in the */usr/lib* directory.

Note Some older Driver Manager packages do not create the soft link from *libodbc.so.1* to *libodbc.so*. Sybase recommends that you manually create this link. The ODBC Driver Manager also includes another shared library called *libodbcinst.so.1*. A soft link from this file to *libodbcinst.so* should also exist. If it is not on your system, you should create one.

In order to link an ODBC application against the ODBC Driver Manager, pass the *-lodbc* flag to the linker.

If the ODBC Driver Manager is not installed in the */usr/lib* directory, you also need to pass the following to the linker:

-Ldir

where *dir* is the directory where the ODBC Driver Manager shared libraries are located.

Mac OS X

The iODBC Driver Manager includes a dynamic library named *libiodbc.dylib*, typically located in the */usr/lib* directory. To link an ODBC application against the iODBC Driver manager, pass the *-liodbc* flag to the linker.

If you use the unixODBC Driver Manager instead of iODBC, the linker flag should be *-lodbc*.

If the ODBC Driver Manager is not installed in the */usr/lib* directory, you also need to pass the following flag to the linker:

-Ldir

where *dir* is the directory where the ODBC Driver Manager shared libraries are located.

Building applications without using an ODBC Driver Manager

You cannot build your applications directly against the ASE ODBC Driver on Windows platforms—you need to build your applications against an ODBC Driver Manager.

You can build applications without using an ODBC Driver Manager on Linux. The ASE ODBC Driver is a shared dynamic library called *libsybdrvodb.so*. This file is usually located in the *\$SYBASE/DataAccess/ODBC/lib* directory for Linux 32-bit, or the *\$SYBASE/DataAccess64/ODBC/lib* directory for Linux 64-bit, where *\$SYBASE* is the Sybase installation root directory.

❖ **To link an ODBC application with the ASE ODBC Driver on Linux**

- 1 Pass the *-lsybdrvodb* and *-L<dir to ASE ODBC Driver>* flags to the linker.
- 2 When deploying your application, verify that the directory containing the ASE ODBC Driver shared library (*\$SYBASE/DataAccess/ODBC/lib* or *\$SYBASE/DataAccess64/ODBC/lib*, depending on your Linux platform) is included in the user's library path (*LD_LIBRARY_PATH* on Linux).

Using the ASE ODBC Driver samples

The samples for the ASE ODBC Driver are located in the *%SYBASE%\DataAccess\ODBC\samples* directory of a Linux 32-bit platform, or the *%SYBASE%\DataAccess64\ODBC\samples* directory of a Linux 64-bit.

Each directory and sample contains a *README* file that contains instructions on building and running the following samples. The list of samples follows:

- *simple*
- *cursors*
- *advanced*

Defining ODBC handles

ODBC applications use a small set of handles to define basic features, such as database connections and SQL statements. A handle is a 32-bit value on 32-bit platforms and a 64-bit value on 64-bit platforms.

The handle types required for ODBC programs are as follows:

Item	Handle type
Environment	SQLHENV
Connection	SQLHDBC

Item	Handle type
Statement	SQLHSTMT
Descriptor	SQLHDESC

The following handles are used in all ODBC applications:

- **Environment** The environment handle provides a global context in which to access data. Every ODBC application must allocate exactly one environment handle upon starting, and must free it at the end.

The following code allocates an environment handle:

```
SQLHENV env;  
SQLRETURN rc;  
rc = SQLAllocHandle( SQL_HANDLE_ENV,  
                    SQL_NULL_HANDLE, &env );
```

- **Connection** A connection is specified by an ODBC driver and a data source. An application can have several connections associated with its environment. Allocating a connection handle does not establish a connection; a connection handle must be allocated first and then used when the connection is established.

The following code allocates a connection handle:

```
SQLHDBC dbc;  
SQLRETURN rc;  
rc = SQLAllocHandle( SQL_HANDLE_DBC, env, &dbc );
```

- **Statement** A statement handle provides access to a SQL statement and any information associated with it, such as result sets and parameters. Each connection can have several statements. Statements are used both for cursor operations (fetching data) and for single statement execution (such as INSERT, UPDATE, and DELETE).

The following code allocates a statement handle:

```
SQLHSTMT stmt; SQLRETURN rc;  
rc = SQLAllocHandle( SQL_HANDLE_STMT, dbc, &stmt );
```

- **Descriptor** A descriptor is a collection of metadata that describes the parameters of a SQL statement or the columns of a result set, as seen by the application or driver. Thus, a descriptor can fill any of four roles:
 - *Application Parameter Descriptor (APD)* – contains information about the application buffers bound to the parameters in an SQL statement, such as their addresses, lengths, and C datatypes.

- *Implementation Parameter Descriptor (IPD)* – contains information about the parameters in a SQL statement, such as their SQL datatypes, lengths, and nullability.
- *Application Row Descriptor (ARD)* – contains information about the application buffers bound to the columns in a result set, such as their addresses, lengths, and C datatypes.
- *Implementation Row Descriptor (IRD)* – contains information about the columns in a result set, such as their SQL datatypes, lengths, and nullability.

The following example illustrates how to retrieve implicitly allocated descriptors:

```
SQLRETURN rc;
SQLHDESC aparamdesc;
SQLHDESC aparamdesc;
SQLHDESC irowdesc;
SQLHDESC arowdesc;
rc = SQLGetStmtAttr(stmt, SQL_ATTR_APP_PARAM_DESC,
                    &aparamdesc, SQL_IS_POINTER);

rc = SQLGetStmtAttr(stmt, SQL_ATTR_APP_ROW_DESC,
                    &arowdesc, SQL_IS_POINTER);

rc = SQLGetStmtAttr(stmt, SQL_ATTR_APP_ROW_DESC,
                    &iparamdesc, SQL_IS_POINTER);

rc = SQLGetStmtAttr(stmt, SQL_ATTR_APP_ROW_DESC,
                    &irowdesc, SQL_IS_POINTER);
```

Implicit descriptors are automatically freed when the statement handle is freed by calling `SQLFreeHandle(SQL_HANDLE_STMT, stmt)`.

Allocating ODBC handles

❖ To allocate an ODBC handle

- 1 Call the `SQLAllocHandle` function, which takes the following parameters:
 - An identifier for the type of item being allocated
 - The handle of the parent item
 - A pointer to the location of the handle to be allocated

For a full description, see `SQLAllocHandle` in the Microsoft *ODBC Programmer's Reference*.

- 2 Use the handle in subsequent function calls.
- 3 Free the object using `SQLFreeHandle`, which takes the following parameters:
 - An identifier for the type of item being freed
 - The handle of the item being freed

For a full description, see `SQLFreeHandle` in the Microsoft *ODBC Programmer's Reference*.

Example

The following code fragment allocates and frees an environment handle:

```
SQLHENV env;
SQLRETURN retcode;
retcode = SQLAllocHandle(SQL_HANDLE_ENV,
                        SQL_NULL_HANDLE, &env );
if ( retcode == SQL_SUCCESS ||
    retcode == SQL_SUCCESS_WITH_INFO )
{
    // success: application code here
}
```

Connecting to a data source

This section describes how to use ODBC functions to establish a connection to an Adaptive Server Enterprise database on a Linux platform.

Note In general, the examples in this chapter use `SQLConnect`.

Choosing an ODBC connection function

ODBC supplies a set of connection functions. Which of the following you use depends on how you expect your application to be deployed and used:

- `SQLConnect`, which is the simplest connection function

SQLConnect takes a data source name (DSN), and an optional user ID and password. You might want to use SQLConnect if you hard-code a data source name into your application.

For more information, see SQLConnect in the Microsoft *ODBC Programmer's Reference*.

- SQLDriverConnect, which connects to a data source using a connection string

SQLDriverConnect allows the application to use Adaptive Server Enterprise-specific connection information that is external to the data source.

Note On Linux, the ASE ODBC Driver supports only SQL_DRIVER_NOPROMPT.

You can also use SQLDriverConnect to connect without specifying a data source.

For more information, see SQLDriverConnect in the Microsoft *ODBC Programmer's Reference*.

- SQLBrowseConnect, which connects to a data source using a connection string, like SQLDriverConnect.

SQLBrowseConnect allows your application to build its own dialog boxes to prompt for connection information, and to browse for data sources used by a particular driver—in this case, the ASE ODBC Driver.

For more information, see SQLBrowseConnect in the Microsoft *ODBC Programmer's Reference*.

For a complete list of connection parameters that can be used in connection strings, see Chapter 2, “Connecting to a Database.”

Establishing a connection

Your application must establish a connection before it can carry out any database operations.

❖ To establish an ODBC connection

- 1 Allocate an ODBC environment:

```
SQLHENV env;
```

```
SQLRETURN retcode;  
retcode = SQLAllocHandle( SQL_HANDLE_ENV,  
                          SQL_NULL_HANDLE, &env );
```

2 Declare the ODBC version.

By declaring that the application follows ODBC version 3, SQLSTATE values and some other version-dependent features are set to the proper behavior. For example:

```
retcode = SQLSetEnvAttr( env,  
                        SQL_ATTR_ODBC_VERSION,  
                        (void*)SQL_OV_ODBC3, 0 );
```

3 If necessary, assemble the data source or connection string.

Depending on your application, you can have a hard-coded data source or connection string, or you can store it externally for greater flexibility.

4 Allocate an ODBC connection handle:

```
retcode = SQLAllocHandle( SQL_HANDLE_DBC, env,  
                          &dbc );
```

5 Set any connection attributes that must be set *before* connecting. (Some connection attributes must be set before establishing a connection, while others can be set either before or after.) For example:

```
retcode = SQLSetConnectAttr( dbc,  
                             SQL_AUTOCOMMIT,  
                             (SQLPOINTER)  
                             SQL_AUTOCOMMIT_OFF,  
                             SQL_IS_INTEGER );
```

6 Call the ODBC connection function:

```
if (retcode == SQL_SUCCESS ||  
    retcode == SQL_SUCCESS_WITH_INFO)  
{  
    printf( "dbc allocated\n" );  
    retcode = SQLConnect( dbc,  
                          (SQLCHAR*) "MANGO", SQL_NTS,  
                          (SQLCHAR*) "sa", SQL_NTS,  
                          (SQLCHAR*) "", SQL_NTS );  
    if (retcode == SQL_SUCCESS ||  
        retcode == SQL_SUCCESS_WITH_INFO)  
    {  
        // successfully connected.  
    }  
}
```

You can find a complete sample of establishing a connection in your installation directory.

Notes on usage

- Every string passed to ODBC has a corresponding length. If the length is unknown, you can pass `SQL_NTS` indicating that it is a Null Terminated String whose end is marked by the null character (`\0`).
- You use the `SQLSetConnectAttr` function to control details of the connection. For example, the following statement turns off ODBC autocommit behavior:

```
retcode = SQLSetConnectAttr( dbc, SQL_AUTOCOMMIT,  
                             (SQLPOINTER) SQL_AUTOCOMMIT_OFF,  
                             SQL_IS_INTEGER );
```

Many aspects of the connection can be controlled through the connection parameters. For more information, see Chapter 2, “Connecting to a Database.”

For more information including a list of connection attributes, see `SQLSetConnectAttr` in the Microsoft *ODBC Programmer's Reference*.

Using threads and connections in ODBC applications

You can develop multithreaded ODBC applications for Adaptive Server Enterprise. Sybase recommends that you use a separate connection for each thread. However, you are allowed to share an open connection among multiple threads.

Executing SQL statements

ODBC includes several functions for executing SQL statements:

- **Direct execution** ASE parses the SQL statement, prepares an access plan, and executes the statement. Parsing and access plan preparation are called preparing the statement.
- **Bound parameter execution** You can construct and execute a SQL statement using bound parameters to set values for statement parameters at runtime. Bind parameters are also used with prepared statements to provide performance benefits for statements that are executed more than once.

- **Prepared execution** The statement preparation is carried out separately from the execution. For statements that are to be executed repeatedly, this avoids repeated preparation and as a result improves performance.

Executing statements directly

The `SQLExecDirect` function prepares and executes a SQL statement. Optionally, the statement can include parameters.

The following code fragment illustrates how to execute a statement without parameters. The `SQLExecDirect` function takes a statement handle, a SQL string, and a length or termination indicator, which in this case is a null-terminated string indicator.

❖ To execute a SQL statement in an ODBC application

- 1 Allocate a handle for the statement using `SQLAllocHandle`.

For example, the following statement allocates a `SQL_HANDLE_STMT` handle with the name “stmt,” on a connection with a handle named “dbc”:

```
SQLAllocHandle( SQL_HANDLE_STMT, dbc, &stmt );
```

- 2 Call the `SQLExecDirect` function to execute the statement.

For example, the following lines declare a statement and execute it:

```
SQLCHAR *deletestmt =  
    "DELETE FROM department WHERE dept_id = 201";  
SQLExecDirect( stmt, deletestmt, SQL_NTS );
```

For more information, see `SQLExecDirect` in the Microsoft *ODBC Programmer's Reference*.

Executing statements with bound parameters

This section describes how to construct and execute a SQL statement, using bound parameters to set values for statement parameters at runtime.

❖ To execute a SQL statement with bound parameters in an ODBC application

- 1 Allocate a handle for the statement using `SQLAllocHandle`.

For example, the following statement allocates a `SQL_HANDLE_STMT` handle the with name “stmt”, on a connection with a handle named “dbc”:

```
SQLAllocHandle( SQL_HANDLE_STMT, dbc, &stmt );
```

2 Bind parameters for the statement using `SQLBindParameter`.

For example, the following lines declare variables to hold the values for the department ID, department name, and manager ID, as well as for the statement string itself. Then, they bind parameters to the first, second, and third parameters of a statement executed using the “stmt” statement handle.

```
#defined DEPT_NAME_LEN 20

SQLINTEGER cbDeptID = 0,
    cbDeptName = SQL_NTS, cbManagerID = 0;
SQLCHAR deptname[ DEPT_NAME_LEN ];
SQLSMALLINT deptID, managerID;
SQLCHAR *insertstmt =
    "INSERT INTO department "
    "( dept_id, dept_name, dept_head_id )"
    "VALUES (?, ?, ?,)";
SQLBindParameter( stmt, 1, SQL_PARAM_INPUT,
    SQL_C_SSHORT, SQL_INTEGER, 0, 0,
    &deptID, 0, &cbDeptID);
SQLBindParameter( stmt, 2, SQL_PARAM_INPUT,
    SQL_C_CHAR, SQL_CHAR, DEPT_NAME_LEN, 0,
    deptname, 0,&cbDeptName);
SQLBindParameter( stmt, 3, SQL_PARAM_INPUT,
    SQL_C_SSHORT, SQL_INTEGER, 0, 0,
    &managerID, 0, &cbManagerID);
```

3 Assign values to the parameters.

For example, the following lines assign values to the parameters for the fragment of step 2:

```
deptID = 201;
strcpy( (char * ) deptname, "Sales East" );
managerID = 902;
```

Usually, these variables are set in response to user action.

4 Execute the statement using `SQLExecDirect`.

For example, the following line executes the statement string held in “insertstmt” on the “stmt” statement handle.

```
SQLExecDirect( stmt, insertstmt, SQL_NTS) ;
```

Bind parameters are also used with prepared statements to provide performance benefits for statements that are executed more than once.

For more information, see `SQLExecDirect` in the Microsoft *ODBC Programmer's Reference*.

Executing prepared statements

The ASE ODBC Driver provides a full set of functions for using prepared statements that provide performance advantages for statements that are used repeatedly.

❖ To execute a prepared SQL statement

- 1 Prepare the statement using `SQLPrepare`.

For example, the following code fragment illustrates how to prepare an insert statement:

```
SQLRETURN    retcode;
SQLHSTMT     stmt;
retcode = SQLPrepare( stmt,
                      "INSERT INTO department"
                      "( dept_id, dept_name, dept_head_id )"
                      "VALUES (?, ?, ?,)",
                      SQL_NTS);
```

where:

- *retcode* holds a return code that should be tested for success or failure of the operation.
- *stmt* provides a handle to the statement.
- *?* is a statement parameter marker.

- 2 Set statement parameter values using `SQLBindParameter`.

For example, the following function call sets the value of the *dept_id* variable:

```
SQLBindParameter( stmt,
                  1,
                  SQL_PARAM_INPUT,
                  SQL_C_SHORT,
                  SQL_INTEGER,
                  0,
                  0,
                  &sDeptID,
                  0,
                  &cbDeptID);
```

where:

- *stmt* is the statement handle.
- *1* indicates that this call sets the value of the first parameter.
- *SQL_PARAM_INPUT* indicates that the parameter is an input statement.
- *SQL_C_SHORT* indicates the C datatype being used in the application.
- *SQL_INTEGER* indicates the SQL datatype being used in the database.
- *0* indicates the column precision.
- *0* indicates the number of decimal digits.
- *&sDeptID* is a pointer to a buffer for the parameter value.
- *0* indicates the length of the buffer, in bytes.
- *&cbDeptID* is a pointer to a buffer for the length of the parameter value.

3 Bind the other two parameters and assign values to sDeptId:

```
SQLBindParameter( stmt, 2, SQL_PARAM_INPUT,  
                  SQL_C_CHAR, SQL_CHAR, DEPT_NAME_LEN, 0,  
                  deptname, 0, &cbDeptName);
```

```
SQLBindParameter( stmt, 3, SQL_PARAM_INPUT,  
                  SQL_C_SSHORT, SQL_INTEGER, 0, 0,  
                  &managerID, 0, &cbManagerID);
```

4 Execute the statement:

```
retcode = SQLExecute( stmt);
```

You can repeat steps 2 through 4 multiple times.

5 Drop the statement using SQLFreeHandle.

Dropping the statement frees resources associated with the statement itself.

Working with result sets

ODBC applications use cursors to manipulate and update result sets. The ASE ODBC Driver provides extensive support for different kinds of cursors and cursor operations.

Choosing cursor characteristics

ODBC functions that execute statements and manipulate result sets use cursors to carry out their tasks. Applications open a cursor implicitly when they execute a statement that returns a result set.

For applications that move through a result set only in a forward direction and do not update the result set, cursor behavior is relatively straightforward. By default, ODBC applications request this behavior. ODBC defines a read-only, forward-only cursor, and the ASE ODBC Driver provides a cursor optimized for performance in this case.

To set the required ODBC cursor characteristics, call the `SQLSetStmtAttr` function that defines statement attributes. You must call `SQLSetStmtAttr` before executing a statement that returns a result set.

You can use `SQLSetStmtAttr` to set many cursor characteristics. The characteristic that determines the cursor type for the ASE ODBC Driver is `SQL_ATTR_CONCURRENCY`. You can set one of the following values:

- **SQL_CONCUR_READ_ONLY** Disallow updates. This is the default.
- **SQL_CONCUR_LOCK** Use the lowest level of locking needed to verify that the row can be updated.

For more information, see `SQLSetStmtAttr` in the Microsoft *ODBC Programmer's Reference*.

Example

The following fragment requests an updateable cursor:

```
SQLAllocHandle( SQL_HANDLE_STMT, dbc, &stmt );
SQLSetStmtAttr( stmt, SQL_ATTR_CONCURRENCY,
                SQL_CONCUR_LOCK, 0 );
```

Note Before using cursors, verify that `UseCursor` property is set to 1. The default value for `UseCursor` is 0.

Retrieving data

To retrieve rows from a database, you execute a `select` statement using `SQLExecute` or `SQLExecDirect`. This opens a cursor on the statement. Then, use `SQLFetch` or `SQLFetchScroll` with `SQL_FETCH_NEXT` option to fetch rows through the cursor. When an application frees the statement using `SQLFreeStmt` with `SQL_CLOSE` option, it closes the cursor.

To fetch values from a cursor, your application can use either `SQLBindCol` or `SQLGetData`:

- If you use `SQLBindCol`, values are automatically retrieved on each fetch.
- If you use `SQLGetData`, you must call it for each column after each fetch.

`SQLGetData` is used to fetch values in pieces for columns such as `LONG VARCHAR` or `LONG BINARY`. As an alternative, you can set the `SQL_ATTR_MAX_LENGTH` statement attribute to a value large enough to hold the entire value for the column. For `SQL_ATTR_MAX_LENGTH`, the default value is 32KB.

The following code fragment from the *simple* sample opens a cursor on a query and retrieves data through the cursor. Error checking has been omitted to make the example easier to read.

```
SQLExecDirect( stmt, "select au_fname from authors ", SQL_NTS );
retcode = SQLBindCol( stmt, 1, SQL_C_CHAR, aufName,
                     sizeof(aufName), &aufNameLen);
while( retcode == SQL_SUCCESS
      || retcode == SQL_SUCCESS_WITH_INFO)
{
    retcode = SQLFetch( stmt );
}
```

Updating and deleting rows through a cursor

To open a cursor for updates or deletes, you can set a statement attribute called `SQL_ATTR_CONCURRENCY` to `SQL_CONCUR_LOCK`:

```
SQLSetStmtAttr( stmt, SQL_ATTR_CONCURRENCY, (SQLPOINTER)
               SQL_CONCUR_LOCK, 0 );
```

The following code fragment from the *cursor* sample illustrates using cursors for updates and deletes. Error checking has been omitted for clarity.

```
/* Set statement attribute for an updateable cursor */
SQLSetStmtAttr( stmt, SQL_ATTR_CONCURRENCY,
```

```
(SQLPOINTER)SQL_CONCUR_LOCK, 0);
SQLSetCursorName(stmt1, "CustUpdate", SQL_NTS);
SQLExecDirect(stmt1, "select LastName from t_CursorTable ",
    SQL_NTS) ;
SQLFetch(stmt1);
SQLExecDirect(stmt2, "Update t_CursorTable"
    "set LastName='UpdateLastName'"
    "where current of CustUpdate",
    SQL_NTS) ;
```

For the complete code, refer to the *cursor.cpp* sample.

Using scrollable cursors

Scrollable cursors can go backward as well as forward to more easily support screen-based applications. When a user scrolls backward and forward, the back end provides the corresponding data.

Setting the UseCursor connection property

To determine whether client-side or server-side scrollable cursors are used, you must set the UseCursor property:

- When the UseCursor connection property is set to 1, server-side scrollable cursors are used if ASE version is 15.0 or later. In earlier versions of the ASE server, server-side scrollable cursors are not available.
- When the UseCursor connection property is set to 0, client-side scrollable cursors (cached result sets) are used, regardless of the ASE version.

Warning! Using client-side scrollable cursors is resource-intensive.

Support for the Static Insensitive scrollable cursor

The ASE ODBC Driver supports the Static Insensitive scrollable cursor. It implements the ODBC SQLFetchScroll method to scroll and fetch rows. The SQLFetchScroll method is a standard ODBC method defined in *Microsoft Open Database Connectivity Software Development Kit Programmer's Reference, Volume 2*, which is part of the MSDN library. Go to the Microsoft Web site at <http://msdn.microsoft.com/en-us/library/ms714177.aspx> for more information.

The ODBC driver supports the following scrolling types:

- `SQL_FETCH_NEXT` – return the next rowset.
- `SQL_FETCH_PRIOR` – return the prior rowset.
- `SQL_FETCH_RELATIVE` – return the rowset *n* from the start of the current rowset.
- `SQL_FETCH_FIRST` – return the first rowset in the result set.
- `SQL_FETCH_LAST` – return the last complete rowset in the result set.
- `SQL_FETCH_ABSOLUTE` – return the rowset starting at row *n*.

Setting scrollable cursor attributes

You must set the following attributes to use scrollable cursors:

- `SQL_ATTR_CURSOR_SCROLLABLE` – the type of scrollable cursor you are using. It should be set to the value of `SQL_SCROLLABLE`. Possible values are static, semi-sensitive, and insensitive.
- `SQL_ATTR_CURSOR_SENSITIVITY` – the sensitivity value for this scrollable cursor.

Note The only supported value for this is `SQL_INSENSITIVE`.

The following are *optional* attributes when using scrollable cursors:

- `SQL_ATTR_ROW_ARRAY_SIZE` – the number of rows that you want returned from each call to the `SQLFetchScroll()` method.

Note If you do not set this value, the default value of one row is used.

- `SQL_ATTR_CURSOR_TYPE` – The type of scrollable cursor you are using.

Note The only supported values for this are `SQL_CURSOR_FORWARD_ONLY` or `SQL_CURSOR_STATIC`.

- `SQL_ATTR_ROWS_FETCHED_PTR` – the address where the number of rows fetched are stored. The `SQL_ATTR_ROWS_FETCHED_PTR` points to a variable of datatype `SQLINTEGER`.
- `SQL_ATTR_ROW_STATUS_PTR` – the address where the row status is stored. The `SQL_ATTR_ROW_STATUS_PTR` points to a variable of datatype `SQLUSMALLINT`.

Executing scrollable cursors

❖ To set up a program to execute a scrollable cursor

- 1 Set the scrollable cursor attributes for your environment.

See “Setting scrollable cursor attributes” on page 20 for more information.

- 2 Bind the results. For example, add the following to your program:

```
res=SQLBindCol(m_StatementHandle, 2, SQL_C_DOUBLE, price, 0, NULL);
res=SQLBindCol(m_StatementHandle, 3, SQL_C_LONG, quantity, 0, NULL);
```

- 3 Scroll and fetch by using SQLFetchScroll(). For example, add the following to your program:

```
res = SQLSetStmtAttr(m_StatementHandle,
SQL_ATTR_CURSOR_SCROLLABLE ,
(SQLPOINTER)SQL_SCROLLABLE,SQL_IS_INTEGER);

res = SQLSetStmtAttr(m_StatementHandle,
SQL_ATTR_CURSOR_SENSITIVITY , (SQLPOINTER)SQL_INSENSITIVE,
SQL_IS_INTEGER);

res = SQLFetchScroll(m_StatementHandle, SQL_FETCH_NEXT,0);
res = SQLFetchScroll(m_StatementHandle, SQL_FETCH_PRIOR,0);
res = SQLFetchScroll(m_StatementHandle, SQL_FETCH_FIRST,0);
res = SQLFetchScroll(m_StatementHandle, SQL_FETCH_LAST,0);
res = SQLFetchScroll(m_StatementHandle, SQL_FETCH_ABSOLUTE,2);
res = SQLFetchScroll(m_StatementHandle, SQL_FETCH_ABSOLUTE,-2);
res = SQLFetchScroll(m_StatementHandle, SQL_FETCH_RELATIVE,1);
```

- 4 Execute the Select statement. For example, add the following to your program:

```
res = SQLExecDirect(m_StatementHandle, (SQLCHAR "select price, quantity
from book" SQL_NTS);
```

- 5 Close the result set and the cursor. For example, add the following to your program:

```
res = SQLFreeStmt(m_StatementHandle,SQL_CLOSE);
```

Looking at results

After you execute a scrollable cursor, you will see these results, assuming a total of N rows and a rowset m where $N > m$:

Result	Interpretation
Absolute 0	No row is returned, error.
Absolute 1	m row is returned.
Absolute N	1 row is returned.
Absolute $N+1$	No row is returned, error.
First	The first $(1..m)$ rows are returned.
Last	The last $(N-m+1 .. N)$ rows are returned.
Next	The same as SQLFetch().
Prior	Return the rowset that is before current rowset.

The following results are expected if the current cursor points to row k and $k-a > 0$, $k+m+a < N$, $a \neq 0$:

Result	Interpretation
Relative $-a$	The rows $(k-a, k-a+m-1)$ are returned.
Relative a	The rows $(k+a, k+a+m-1)$ are returned.

Implicit setting of scrolling cursor attributes

Certain attributes are set implicitly when your application sets specific attributes. The supported ODBC scrollable cursor attributes set implicitly are as follows:

Application sets attribute to	Other attributes set implicitly
SQL_ATTR_CONCURRENCY to SQL_CONCUR_READ_ONLY	SQL_ATTR_CURSOR_SENSITIVITY to SQL_INSENSITIVE
SQL_ATTR_CONCURRENCY to SQL_CONCUR_LOCK	SQL_ATTR_CURSOR_SENSITIVITY to SQL_SENSITIVE
SQL_ATTR_CURSOR_SCROLLABLE to SQL_NONSCROLLABLE	SQL_ATTR_CURSOR_TYPE to SQL_CURSOR_FORWARD_ONLY
SQL_ATTR_CURSOR_SENSITIVITY to SQL_INSENSITIVE	SQL_ATTR_CONCURRENCY to SQL_CONCUR_READ_ONLY SQL_ATTR_CURSOR_TYPE to SQL_CURSOR_STATIC
SQL_ATTR_CURSOR_TYPE to SQL_CURSOR_FORWARD_ONLY	SQL_ATTR_CURSOR_SCROLLABLE to SQL_NONSCROLLABLE
SQL_ATTR_CURSOR_TYPE to SQL_CURSOR_STATIC	SQL_ATTR_CURSOR_SCROLLABLE to SQL_SCROLLABLE

Calling stored procedures

This section describes how to create and call stored procedures, and how to process the results from an ODBC application.

For a full description of stored procedures and triggers, see the *ASE Reference Manual*.

Procedures and result sets

There are two types of procedures: those that return result sets, and those that do not. You can use `SQLNumResultCols` to tell the difference: The number of result columns is zero if the procedure does not return a result set. If there is a result set, you can fetch the values using `SQLFetch` or `SQLFetchScroll` just like any other cursor.

Pass parameters to procedures using parameter markers (question marks). Use `SQLBindParameter` to assign a storage area for each parameter marker, whether it is an *INPUT*, *OUTPUT*, or *INOUT* parameter.

Example

The *advanced* sample illustrates a stored procedure that returns an output parameter and a return value, and another stored procedure that returns multiple result sets. Error checking has been omitted to make the example easier to read.

```
/*
Example 1: How to call a stored procedure and use input and output parameters
*/

SQLBindParameter(stmt, 1, SQL_PARAM_OUTPUT, SQL_C_SLONG,
                 SQL_INTEGER, 0, 0, &retVal, 0,
                 SQL_NULL_HANDLE);
SQLBindParameter(stmt, 2, SQL_PARAM_INPUT, SQL_C_CHAR,
                 SQL_CHAR, 4, 0, stor_id, sizeof(stor_id),
                 SQL_NULL_HANDLE);
SQLBindParameter(stmt, 3, SQL_PARAM_OUTPUT, SQL_C_CHAR,
                 SQL_VARCHAR, 20, 0, ord_num, sizeof(ord_num),
                 &ordnumLen);
SQLBindParameter(stmt, 4, SQL_PARAM_INPUT, SQL_C_CHAR,
                 SQL_VARCHAR, 40, 0, date, sizeof(date),
                 &dateLen);

SQLExecDirect(stmt, "{ ? = call sp_selectsales(?,?,?) }", SQL_NTS);

/*

At this point retVal contains the return value as returned from
the stored procedure and the ord_num contains the order number
```

```
as returned from the stored procedure
*/

/*
Example 2: How to call stored procedures returning multiple result sets
*/

SQLBindParameter(stmt, 1, SQL_PARAM_INPUT, SQL_C_CHAR,
    SQL_CHAR , 4, 0, stor_id, sizeof(stor_id) ,
    SQL_NULL_HANDLE);

SQLExecDirect(stmt, "{ call sp_multipleresults(?) }", SQL_NTS);

SQLBindCol( stmt, 1, SQL_C_CHAR, dbValue, sizeof(dbValue),
    &dbValueLen);

SQLSMALLINT count = 1;

while(retcode == SQL_SUCCESS
    || retcode == SQL_SUCCESS_WITH_INFO)
{
    retcode = SQLFetch( stmt );
    if (retcode == SQL_NO_DATA)
    {
        /*
        -- End of first result set --
        */
        if(count == 1)
        {
            retcode = SQLMoreResults(stmt);
            count ++;
        }
        /*
        At this point dbValue contains the value in the current row of the
            result
        */
    }
}
```

Handling errors

Errors in ODBC are reported using the return value from each of the ODBC function calls and either the `SQLGetDiagField` function or the `SQLGetDiagRec` function. The `SQLERROR` function was used in ODBC versions up to, but not including, version 3. As of version 3, the `SQLERROR` function has been replaced by the `SQLGetDiagRec` and `SQLGetDiagField` functions.

Every ODBC function returns a `SQLRETURN` that is one of the following status codes:

Status code	Description
<code>SQL_SUCCESS</code>	No error.
<code>SQL_SUCCESS_WITH_INFO</code>	The function completed, but a call to <code>SQLGetDiagRec</code> will indicate a warning. The most common cause for this status is that a value being returned is too long for the buffer provided by the application.
<code>SQL_INVALID_HANDLE</code>	An invalid environment, connection, or statement handle was passed as a parameter. This often happens if a handle is used after it has been freed, or if the handle is the null pointer.
<code>SQL_NO_DATA</code>	There is no information available. The most common use for this status is when fetching from a cursor; it indicates that there are no more rows in the cursor.
<code>SQL_NEED_DATA</code>	Data is needed for a parameter. This is an advanced feature described in the ODBC Software Development Kit documentation under <code>SQLParamData</code> and <code>SQLPutData</code> .

Every environment, connection, and statement handle can have one or more errors or warnings associated with it. Each call to `SQLGetDiagRec` returns the information for one error and removes the information for that error. If you do not call `SQLGetDiagRec` to remove all errors, the errors are removed on the next function call that passes the same handle as a parameter.

Each call to `SQLGetDiagRec` can pass either an environment, connection, or statement handle. The first call passes in a handle of type `SQL_HANDLE_DBC` to get the error associated with a connection. The second call passes in a handle of type `SQL_HANDLE_STMT` to get the error associated with the statement that was just executed.

SQLGetDiagRec returns SQL_SUCCESS if there is an error to report (*not* SQL_ERROR), and SQL_NO_DATA_FOUND if there are no more errors to report.

Example 1

The following code fragments use SQLGetDiagRec and return codes:

```
retcode = SQLAllocHandle(SQL_HANDLE_STMT, dbc, &stmt );
if( retcode == SQL_ERROR )
{
    SQLGetDiagRec(SQL_HANDLE_DBC,dbc, 1, NULL,NULL,
        errmsg, 100, NULL);
    /* Assume that print_error is defined */
    print_error( "Allocation failed", errmsg );
    return;
}
```

Example 2

```
retcode = SQLExecDirect( stmt,
    "delete from sales_order_items where id=2015",
    SQL_NTS );
if( retcode == SQL_ERROR )
{
    SQLGetDiagRec(SQL_HANDLE_STMT,stmt, 1, NULL,NULL,
        errmsg, 100, NULL);
    /* Assume that print_error is defined */
    print_error( "Failed to delete items", errmsg );
    return;
}
```

Datatype mappings

The following table describes the ASE ODBC Driver datatype mappings.

Table 1-1: Datatype mappings

ASE datatype	ODBC SQL datatype	ODBC bind datatype
bigint	SQL_BIGINT	SQL_C_BIGINT
binary	SQL_BINARY	SQL_C_BINARY
bit	SQL_BIT	SQL_C_BIT
char	SQL_CHAR	SQL_C_CHAR
date	SQL_TYPE_DATE	SQL_C_TYPE_DATE or SQL_C_CHAR

ASE datatype	ODBC SQL datatype	ODBC bind datatype
datetime	SQL_TYPE_TIMESTAMP	SQL_C_TYPE_TIMESTAMP or SQL_C_CHAR
decimal	SQL_DECIMAL	SQL_C_NUMERIC or SQL_C_CHAR
double	SQL_DOUBLE	SQL_C_DOUBLE
float(<16)	SQL_REAL	SQL_C_FLOAT
float(>=16)	SQL_DOUBLE	SQL_C_DOUBLE
image	SQL_LONGVARIABLE	SQL_C_BINARY
int[eger]	SQL_INTEGER	SQL_C_LONG
money	SQL_DECIMAL	SQL_C_NUMERIC or SQL_C_CHAR
nchar	SQL_CHAR	SQL_C_CHAR
nvarchar	SQL_VARCHAR	SQL_C_CHAR
numeric	SQL_NUMERIC	SQL_C_NUMERIC or SQL_C_CHAR
real	SQL_REAL	SQL_C_FLOAT
smalldatetime	SQL_TYPE_TIMESTAMP	SQL_C_TYPE_TIMESTAMP or SQL_C_CHAR
smallint	SQL_SMALLINT	SQL_C_SHORT
smallmoney	SQL_DECIMAL	SQL_C_NUMERIC or SQL_C_CHAR
text	SQL_LONGVARCHAR	SQL_C_CHAR
time	SQL_TYPE_TIME	SQL_C_TYPE_TIME or SQL_C_CHAR
timestamp	SQL_BINARY	SQL_C_BINARY
tinyint	SQL_TINYINT	SQL_C_TINYINT
unichar	SQL_WCHAR	SQL_C_CHAR
unitext	SQL_WLONGVARIABLE	SQL_C_CHAR
univarchar	SQL_WVARCHAR	SQL_C_CHAR
unsignedbigint	SQL_BIGINT	SQL_C_UBIGINT
unsignedint	SQL_INTEGER	SQL_C_ULONG
unsignedsmallint	SQL_SMALLINT	SQL_C_USHORT
varbinary	SQL_VARBINARY	SQL_C_BINARY
varchar	SQL_VARCHAR	SQL_C_CHAR

Special instructions for
unichar, varchar, and
unitext

When you use the ASE datatypes `unichar`, `univarchar`, and `unitext`, and then bind either of them to `SQL_C_CHAR`, in Linux, the ASE ODBC Driver needs to convert the data from Unicode to multibyte and vice versa. For this conversion, it needs to have the SYBASE charsets installed in the `$SYBASE` directory. The installation program for Linux includes an option to install these charset files.

Note If the driver does not find the charsets, or if the `$SYBASE` environment variable is not set, then an appropriate error is propagated to the application. To install the SYBASE charsets, you must reinstall the ODBC Driver. See the Software Developer's Kit and Open Server 12.5.1 *Installation Guide* for installation information.

Using computed columns

The ASE Drivers support computed columns that allow you to create a shorthand term for an expression, such as “Pay” for “Salary + Commission,” and to make that column indexable, as long as its datatype can be indexed. Computed columns are defined by an expression, whether from regular columns in the same row, functions, arithmetic operators, and path names, including their metadata information.

Using server-specified packet size

Clients and servers have to be prepared to reserve memory to store the packages used for communication between them. These packages are called Protocol Data Units, or PDUs. Every PDU starts with an 8-byte header containing a 2-byte, unsigned integer describing the actual size in bytes of the current PDU (including the header itself). Clients and servers must know the maximum size that a PDU sent by the other party could be, and this is called the *packet size*. The packet size is negotiated at login time.

When connected to ASE 15.0, the ODBC Driver lets the server select the packet size to optimize performance. When connected to an pre-15.0 ASE server, the ODBC Driver uses 512 as the packet size, unless you specify the `packetsize` property. If you do not want the server to decide the packet size, you need to set `EnableServerPacketSize` to 0. If you have memory restrictions, you need to set `RestrictMaximumPacketSize` to a number (in multiples of 512) so that ASE and the ODBC Driver will not negotiate a packet size greater than the one you specified.

Using Large Identifiers for database objects

The ASE Drivers support the new ASE large identifiers, or names, for database objects. Some object names in ASE 15.0 now have new limits of 255 bytes. As a result, you can now have longer names for objects such as tables, columns, and procedures.

Warning! If you use large identifiers in C++ programs or client applications, you must allocate sufficient buffer lengths to avoid data truncation.

Connecting to a Database

This chapter describes how client applications connect to Sybase Adaptive Server Enterprise using ODBC.

Topic	Page
Introduction to connections	31
How connection parameters work	32
Character sets	33
Configuring the ASE ODBC Driver	34
Connecting using a data source	40

Introduction to connections

Any client application that uses Adaptive Server Enterprise must establish a connection to the Adaptive server before any work can be done. The connection forms a channel through which all activity from the client application takes place. For example, your user ID determines permissions to carry out actions on the database—and the database server has your user ID because it is part of the request to establish a connection. The ASE ODBC Driver uses connection information included in the call from the client application (perhaps together with information held on disk in an initialization file) to locate and connect to an ASE server running the required database.

Installing ODBC MetaData stored procedures

You must install the ODBC MetaData stored procedures on any Adaptive Servers that you want to connect to using the ODBC Driver.

Windows

❖ To install the stored procedures on a target Adaptive Server on Windows

- 1 Change to the *sp* directory under the ODBC installation directory.

- 2 Execute the *install_odbc_sprocs* script:

```
install_odbc_sprocs ServerName username [password]
```

where:

- **ServerName** is the name of the Adaptive Server.
- **username** is the user name to connect to the server.
- **[password]** is the password for the user name. If the value is null, leave the parameter empty.

Linux

❖ To install the stored procedures on a target Adaptive Server on Linux

- 1 Change to the *sp* directory under the ODBC installation directory.

- 2 Execute the *install_odbc_sprocs* script:

```
./install_odbc_sprocs ServerName username  
[password]
```

where:

- **ServerName** is the name of the Adaptive Server.
- **username** is the user name to connect to the server.
- **[password]** is the password for the user name. If the value is null, leave the parameter empty.

How connection parameters work

When an application connects to a database, it uses a set of connection parameters to define the connection. Connection parameters include information such as the server name, the database name, and a user ID. A keyword-value pair (of the form *parameter=value*) specifies each connection parameter. For example, you specify the user ID connection parameter as follows:

```
UID=sa
```

Connection
parameters passed as
connection strings

Connection parameters are passed to the ASE ODBC driver as a connection string and are separated by semicolons:

```
parameter1=value1;parameter2=value2;...
```

In general, the connection string built by an application and passed to the driver does not correspond directly to the way a user enters the information. Instead, a user can fill in a dialog box, or the application can read connection information from an initialization file.

Character sets

The ASE server can be configured to store character data in a specific charset. When the ODBC Driver connects to the ASE server, the server and the driver negotiate a charset. Then, the driver and the server exchange multibyte data in the negotiated charset. The ASE ODBC driver determines the charset depending on the platform:

- On Windows, the ASE ODBC Driver by default negotiates the same default charset as the ASE server. The default charset is ServerDefault. However, if you want to use the ClientDefault charset, you need to specify the value for the CodePageType property, which can be ANSI or OEM. The default is ANSI. You can also specify a User specified charset by specifying a valid ASE charset for the Charset connection property.
- On Linux, the ASE ODBC Driver by default examines the LC_CTYPE and LANG environment variables. If they are not set, it defaults to ISO 8859-1. If one of these environment variables are set, it then looks for *locales.dat* in the *\$SYBASE/locales/locales.dat* directory to pickup the corresponding ASE charset. If the file is not found, it then looks into its own map in memory to lookup the corresponding ASE charset.

During login, the driver then negotiates this client charset. This behavior can be overwritten by specifying the CharSet connection property to a valid ASE charset or "ServerDefault." When set to "ServerDefault," the driver behavior is the same as in the Windows platform. You can also specify a User specified charset by specifying a valid ASE charset for CharSet connection property.

Configuring the ASE ODBC Driver

When connecting to the database, ODBC applications typically use ODBC data sources. An ODBC data source is a set of connection parameters, stored in the registry or in a file. ODBC data sources on non-Windows platforms typically reside in an *ini* file. Most ODBC Driver Managers provide a GUI tool to configure ODBC Driver and data sources.

Windows

When you use the Sybase SDK installation program to install the ASE ODBC Driver, it registers the driver on the local machine. You can manually register the ASE ODBC Driver on Windows using the `regsvr32` utility.

Registering the ASE ODBC Driver on Windows

Note You do not need to manually register the ASE ODBC Driver if you have used the Sybase SDK installation program to install ASE ODBC Driver on this machine.

❖ **To register the ASE ODBC Driver manually**

- 1 Change to the `%SYBASE%\DataAccess\ODBC\dll` directory, which contains the ASE ODBC Driver DLL.
- 2 Run the `regsvr32` utility to create registry entries in the `HKEY_LOCAL_MACHINE\SOFTWARE\ODBC\ODBCINST.INI` key:

```
regsvr32 sybdrvodb.dll
```

Configuring a data source on Windows

❖ **To configure a data source**

- 1 Launch the ODBC Administrator. See the online help for your specific Windows operating system for detailed instructions.
- 2 Select the User DSN tab. Click Add.
- 3 Choose “Adaptive Server Enterprise” from the list of drivers.
- 4 Click Finish.

- 5 Select the General tab. Enter values in the following fields:
 - Data Source Name: a name for your data source
 - Description: a description for your data source
 - Server Name: an Adaptive Server Enterprise host name
 - Server Port: an Adaptive Server Enterprise port number
 - Database Name: a database name
 - Logon ID: a user name to login to the Adaptive Server Enterprise database
- 6 Select Use Cursors if you want cursors to be opened for every select statement.
- 7 Complete the Connection and Advanced tabs as needed.
- 8 Click OK to save the changes.

Note For a detailed explanation of connection parameters, see “Using connection parameters” on page 40.

Linux

The unixODBC Driver Manager supports configuring drivers and data sources from a GUI as well as the command line. Refer to the ODBC Driver Manager's documentation for instructions on the GUI tool and command line syntax.

Note The ASE ODBC Driver and data sources that use this driver cannot be configured using the GUI tools from the unixODBC Driver Manager. You must use the command line interface.

When configuring the driver and data sources using the unixODBC Driver Manager command line tool, you must supply a template file. Sample templates are described in the following section. You can also find these templates in the `$SYBASE/DataAccess/ODBC/samples` directory of Linux 32-bit platforms, or in the `$SYBASE/DataAccess64/ODBC/samples` of Linux 64-bit platforms.

The following is an example of a driver template file:

```
[Adaptive Server Enterprise]
```

```
Description=Sybase ODBC Driver
Driver=/install dir/driver library name
FileUsage=-1
```

where:

- *install dir* is the path to the ASE ODBC Driver installation.
- *driver library name* is the name of the driver library.

Installing the ASE ODBC Driver on Linux

To install the ASE ODBC Driver, execute the following command to install the ASE ODBC Driver:

```
# odbcinst -i -d -f driver template file
```

where *driver template file* is the complete path to the ASE ODBC Driver template file.

Note In most cases, this command needs to be executed as the root user because it modifies the *odbcinst.ini* file that is owned by root.

Configuring a data source on Linux

The following is a data source template:

```
[sampledsn]
Description=Sybase ODBC Data Source
UserID=sa
Password=
Driver=Adaptive Server Enterprise
Server=sampleserver
Port=4100
Database=pubs2
UseCursor=1
```

To configure a data source using the unixODBC Driver Manager command line tool, execute the following command to configure a data source for the ASE ODBC Driver using the unixODBC command line tool:

```
# odbcinst -i -s -f dsn template file
```


where *dsn template file* is the complete path to the ASE ODBC Data source template file. This creates entries for the data source in the *odbc.ini* file.

Note The exact command you need to configure ODBC data sources depends on the ODBC Driver Manager you are using.

Mac OS X

During the ASE ODBC Driver installation, the ASE ODBC Driver is configured in the */Library/ODBC/odbcinst.ini* file. To configure the ASE ODBC Driver manually, use the iODBC ODBC Administrator, as described in the following procedure.

Manually configuring the ODBC Driver on Mac OS X

❖ **To use the iODBC Driver Manager**

- 1 Start the iODBC Administrator from Applications | Utilities.
- 2 Select the Drivers tab and click Add.
- 3 In the Description field, enter "Adaptive Server Enterprise" as the Driver description.
- 4 Click Choose to select the installation path in the Driver file field.
You do not need to enter values in the setup file or keyword value pairs fields.
- 5 Click OK to save the changes.

Configuring a data source on Mac OS X

You can configure the ASE ODBC Driver using the iODBC Administrator.

❖ **To configure a data source**

- 1 Start the iODBC Administrator from Applications | Utilities.
- 2 Select the User DSN tab. The Choose a Driver window opens.
- 3 Select the Adaptive Server Enterprise Driver you want to use.
- 4 Click OK.

- 5 Provide a name for your data source in the Data Source Name (DSN) field.
- 6 Provide a description for your data source in the Description field.
- 7 Click Add to add keyword value pairs. Repeat this step until you have added all the keyword value pairs. For example:

Keyword	Value
UserID	sa
Password	
Server	sampleserver
Port	4100
Database	pubs2
UseCursor	1

- 8 Click OK to save the changes.

Note For more information on installing and configuring drivers and data sources using the iODBC Administrator on Mac OS X, look up the iODBC Administrator online help.

ODBC *ini* files

The ODBC Driver Manager stores driver and data source information in *ini* files or the system registry.

Note Refer to your ODBC Driver Manager documentation for the exact path for these *ini* files.

Windows

The *odbc.ini* and *odbcinst.ini* files are located in the *c:\winnt* directory. The Microsoft ODBC Driver Manager looks up these files or the registry at runtime when an application requests a connection to a data source.

Linux

Information about the ODBC Driver installed on the system is saved in the *odbcinst.ini* file. This file is typically located at */etc/odbcinst.ini*.

The information about data sources is saved in one of two files:

- User data source information, available only to that user, is saved in the *\$HOME/.odbc.ini* file, where *\$HOME* is the user home directory.
- System data source information, available to any user on the system, is usually saved in the */etc/odbc.ini* file. If the same data source is defined in both files, the user data source takes precedence.

The ODBC Driver Manager looks up these files at runtime when an application requests a connection to a data source.

Note Refer to your ODBC Driver Manager documentation for the exact path for these *ini* files. Some Driver Manager use alternate locations.

If your application is not using ODBC Driver Manager and uses the ASE ODBC Driver directly, the *ini* file is searched differently: The ASE ODBC Driver first looks for a file named *odbc.ini* in the current working directory; if the file is not found or the data source not found in the file, it looks for *\$SYBASE/odbc.ini*.

Mac OS X

When you use the iODBC ODBC Administrator tool, the *odbcinst.ini* and the *odbc.ini* files are typically located in the */Library/ODBC* directory if the driver or data source was installed system-visible. If the driver or data source was installed to be user-visible, the *odbcinst.ini* and the *odbc.ini* files are in the *\$HOME/Library/ODBC* directory.

At runtime, the iODBC Driver Manager searches for DSN information in *\$HOME/Library/ODBC/odbc.ini*. If your DSN information is in */Library/ODBC/odbc.ini* or in any other location, you need to set an environment variable called ODBCINI to the path to the *odbc.ini* file. For example:

```
setenv ODBCINI full pathname to the odbc.ini file
```

Connecting using a data source

ODBC applications usually use data sources on the client computer for each database you want to connect to. You can store sets of Adaptive Server Enterprise connection parameters as an ODBC data source, in either the system registry or *ini* files. If you have a data source, your connection string can simply name the data source by using the DataSourceName (DSN) connection parameter:

```
DSN=my data source
```

Using connection parameters

Following is a list of connection parameters other than from the DSN parameter that can be supplied to the ASE ODBC Driver.

Table 2-1: Connection parameters

Property names	Description	Required	Default value
UID, UserID	A case-sensitive user ID required to connect to the ASE server.	Yes	Empty
PWD, Password	A case-sensitive password to connect to the ASE server.	No, if the user name does not require a password	Empty
Server	The name or IP address of the ASE server.	Yes	Empty
Port	The port number of ASE server.	Yes	Empty
AnsiNull	Strict ODBC compliance where you cannot use “= NULL.” Instead, you must use “IsNull.”	No	1
ApplicationName	The name used by ASE to identify the client application.	No	Empty
BufferCacheSize	Keeps the input / output buffers in pool. When large results will occur, increase this value to boost performance.	No	20

Property names	Description	Required	Default value
CharSet	<p>The designated character set.</p> <p>On Windows, the ASE ODBC Driver by default negotiates the same default Character set as the ASE server. The default Character set is ServerDefault.</p> <p>On Linux the ASE ODBC Driver uses the Client's Character set based on the environment. If you want to use the ASE server's Character Set, you must specify it in the connection properties with Charset=ServerDefault. The default Character set is ClientDefault. See the section titled "Character sets" on page 33.</p>	No	Empty
ClientHostName	The name of the client host passed in the login record to the server.	No	Empty
ClientHostProc	The identity of client process on this host machine passed in the login record to the server.	No	Empty
CRC	<p>By default, the driver returns the total records updated when multiple update statements are executed in a stored procedure. This count will also include all updates happening as part of the triggers set on an update or an insert.</p> <p>Set this property to 0 if you want the driver to return only the last update count.</p>	No	1
Database	The database to which you want to connect.	No	Empty
DataIntegrity	Enables Kerberos Data Integrity.	No	0 (disabled)
DSPassword	The password used to authenticate on the LDAP server, if the LDAP server does not allow anonymous access. The password can be specified in the Directory Service URL (DSURL) as well.	No	Empty
DSPrincipal	The user name used to authenticate on the LDAP server, if the LDAP server does not allow anonymous access. The principal can be specified in the DSURL as well.	No	Empty
DSURL	The URL to the LDAP server.	No	Empty
DTCProtocol (Windows only)	Allows the driver to use either an XA protocol or OleNative protocol when using distributed transactions. See "Using Distributed Transactions" on page 49, in Chapter 3, "ASE Advanced Features."	No	XA

Property names	Description	Required	Default value
DynamicPrepare	When set to 1, the driver sends SQLPrepare calls to ASE to compile/prepare. This can boost performance if you use the same query repeatedly.	No	0
EnableServerPacketSize	Allows ASE server versions 15.0 or later to choose the optimal packet size.	No	1
EncryptedPassword	<p>Specifies whether password is transmitted in an encrypted format:</p> <ul style="list-style-type: none"> 0 – Use plain text password. 1 – Use encrypted password. If it is not supported, return an error message. 2 – Use encrypted password. If it is not supported, use plain text password. <p>Note When password encryption is enabled, and the server supports asymmetric encryption, this format is used instead of symmetric encryption.</p>	No	0
Encryption	The designated encryption. Possible values: ssl.	No	Empty
FetchArraySize	Specifies the number of rows the driver retrieves when fetching results from the server.	No	25
HASession	Specifies if high availability is enabled. 0 indicates high availability disabled, 1 high availability enabled.	No	0
IgnoreErrorsIfRS Pending	Specifies whether the driver is to continue processing or stop if error messages are present. When set to 1, the driver will ignore errors & continue processing the results if more results are available from the server. When set to 0, the driver will stop processing the results if an error is encountered even if there are results pending	No	0
UseCursor	Specifies whether cursors are to be used by the driver. 0 indicates do not use cursors, and 1 indicates use cursors.	No	0
Language	The language in which ASE returns error messages.	No	Empty – ASE uses English by default

Property names	Description	Required	Default value
LoginTimeout	Number of seconds to wait for a login attempt before returning to the application. If set to 0, the timeout is disabled, and a connection attempt waits for an indefinite period of time.	No	10
MutualAuthentication	Enables Kerberos Mutual Authentication.	No	0 (disabled)
oldpassword	The current password. If oldpassword contains a value that is not null or an empty string, the current password is changed to the value contained in pwd.	No	Empty
pwd	Contains the new password entered by the user. If the oldpassword property does not exist or is null, pwd contains the value of the current password.	No	Empty
PacketSize	The number of bytes per network packet transferred between ASE and the client.	No	Server determined when driver is connected to ASE 15.0 or later. For older ASE servers, the default is 512.
QuotedIdentifier	Specifies if ASE treats character strings enclosed in double quotes as identifiers: <ul style="list-style-type: none"> • 0 = do not enable quoted identifiers • 1 = enable quoted identifiers 	No	0
ReplayDetection	Enables Kerberos Replay Detection.	No	0
RestrictMaximumPacketSize	If the you have memory constraints when EnableServerPacketSize is set to 1, then set this property to an int value in multiples of 512 to a maximum of 65536.	No	0
SecondaryPort	The port number of the ASE server acting as a failover server in an active-active or active-passive setup.	Yes, if HASession is set to 1.	Empty
SecondaryServer	The name or the IP address of the ASE server acting as a failover server in an active-active or active-passive setup.	Yes, if HASession is set to 1.	Empty

Property names	Description	Required	Default value
ServerInitiated Transactions	When SQL_ATTR_AUTOCOMMIT is set to "1" Adaptive Server starts managing transactions as needed. The driver issues a "set chained on" command on the connection. Older ODBC Drivers do not use this feature and manage the job of starting transactions by calling begin tran. Set this property to "0" if you want to maintain the old behavior or require that your connection not use "chained" transaction mode.	No	1
TextSize	The maximum size of binary or text data that will be sent over the wire.	No	Empty – ASE default is 32K.
TightlyCoupled Transaction (Windows only)	When using distributed transactions, if you are using two DSNs which connect to the same ASE server, set this to 1. See "Using Distributed Transactions" on page 49, in Chapter 3, "ASE Advanced Features."	No	0
TrustedFile	If encryption is set to ssl, this property should be set to the path to the Trusted File.	No	Empty

This chapter describes the advanced ASE features you can use with the ASE ODBC Driver.

Topic	Page
Asynchronous execution for ODBC	45
Supported Adaptive Server Cluster Edition features	46
Using Distributed Transactions	49
Using directory services	52
Bookmark and bulk support	55
DSN Migration tool	56
Using password encryption	57
Password expiration handling	59
Using SSL	60
Using failover in high availability systems	65
Enabling Kerberos authentication	69

Asynchronous execution for ODBC

By default, drivers execute ODBC functions synchronously. That is, the application calls a function and the driver returns control to the application when execution is complete. With asynchronous execution, the driver returns control to the application after minimal processing and before execution is complete. This allows the application to execute in parallel other functions while the first function is still executing. Asynchronous execution is beneficial when a task is complex and requires a significant amount of time to execute.

For more information on asynchronous execution and its application, refer to MSDN ODBC Programmer's Reference at <http://msdn.microsoft.com/en-us/library/ms714177.aspx>.

Note The ASE ODBC Driver by Sybase supports a maximum of one concurrent statement in asynchronous mode. Only one concurrent statement, synchronous or asynchronous, can be executed if server-side cursors are used or if the connection's auto-commit is disabled.

To use connection-level asynchronous execution with the ASE ODBC Driver by Sybase, call `SQLSetConnectAttr` and set `SQL_ATTR_ASYNC_ENABLE` to `SQL_ASYNC_ENABLE_ON`.

Note Calling `SQLCancel` when no processing is being done will not close the associated cursors. ODBC applications should explicitly call `SQLFreeStmt` or `SQLCloseCursor` to close cursors.

Supported Adaptive Server Cluster Edition features

This section describes the ASE ODBC Driver features that support the Cluster Edition environment, an environment where multiple Adaptive Servers connect to a shared set of disks and a high-speed private interconnection. This allows Adaptive Server to scale using multiple physical and logical hosts.

For more information about Cluster Edition, see the Adaptive Server Enterprise Cluster Edition *Users Guide to Clusters*.

Login redirection

At any given time, some servers within a Cluster Edition environment are usually more loaded with work than others. When a client application attempts to connect to a busy server, the login redirection feature helps balance the load of the servers by allowing the server to redirect the client connection to less busy servers within the cluster. The login redirection occurs during the login sequence and the client application does not receive notification that it was redirected.

Note When a client application connects to a server that is configured to redirect clients, the login time may increase because the login process is restarted whenever a client connection is redirected to another server.

Connection migration

The connection migration feature allows a server in a Cluster Edition environment to dynamically distribute load, and seamlessly migrate an existing client connection and its context to another server within the cluster. This feature enables the Cluster Edition environment to achieve optimal resource utilization and decrease computing time. Because migration between servers is seamless, the connection migration feature also helps create a truly High Availability (HA), zero-downtime environment.

Note The login redirection and connection migration features are enabled automatically when a client application connects to a server that supports these features.

Note Command execution time may increase during server migration. Sybase recommends that you increase the command timeouts accordingly.

Connection failover in Cluster Edition

Connection failover allows a client application to switch to an alternate Adaptive Server if the primary server becomes unavailable due to an unplanned event, like power outage or a socket failure. In the Adaptive Server Cluster Edition, client applications can failover numerous times to multiple servers using dynamic failover addresses.

With the High Availability enabled, the client application does not need to be configured to know the possible failover targets. Adaptive Server keeps the client updated with the best failover list based on cluster membership, logical cluster usage and load distribution. During failover, the client refers to the ordered failover list while attempting to reconnect. If the driver successfully connects to a server, the driver internally updates the list of host values based on the list returned. Otherwise, the driver throws a connection failure exception.

For more information about connection failover, see “Using failover in high availability systems” on page 65.

Enabling Cluster Edition connection failover

Using the ODBC user interface

One way to enable the Cluster Edition connection failover in ODBC is through the ODBC user interface.

❖ Using the user interface to enable extended failover

- 1 Open the Adaptive Server Enterprise dialog box.
- 2 Go to the Connection tab.
- 3 Select Enable High Availability.
- 4 Optionally, enter the alternate servers and ports in the Alternate Servers field using the format:

```
server1:port1,server2:port2,...,serverN:portN;
```

In establishing a connection, ODBC Driver by Sybase will first try to connect to the primary host and port defined in the General tab of the Adaptive Server Enterprise dialog box. If ODBC fails to establish a connection, ODBC will go through the list of hosts and ports specified in the Alternate Servers field.

Using the ODBC connection string

Another way to enable the connection failover in ODBC is to set the HASession connection string property to 1. You can use SQLDriverConnect to specify a connection string. For example:

```
Driver=AdaptiveServerEnterprise;server=server1;
```

```
port=port1;UID=sa;PWD=;HASession=1;  
AlternateServers=server2:port2,...,serverN:portN;
```

The preceding example defines server1 and port1 as the primary server and port. If ODBC fails to establish connection to the primary server, and alternate servers are defined, it will go through the ordered list of servers and ports specified in the Alternate Servers field until a connection is established or until the end of the list is reached.

Note The list of alternate servers specified in the GUI or the connection string is used only during initial connection. After the connection is established with any available instance, and the client supports high-availability, the client will receive an updated list of the best possible failover targets from the server. This new list overrides the specified list.

Using Distributed Transactions

This section describes how you can use the ASE ODBC driver to participate in two-phase commit transactions. This feature is supported only on Windows and requires that Microsoft Distributed Transaction Coordinator (MS DTC) be the transaction coordinator managing two-phase commit.

Sybase supports all of the following programming models:

- Applications using MS DTC directly
- Applications using Sybase Enterprise Application Server (EAServer, also known as Jaguar)
- Applications using Microsoft Transaction Server (MTS) or COM+

Programming for MS DTC

❖ To program using Microsoft Distributed Transaction Coordinator (MS DTC)

- 1 Connect to MS DTC by using the DtcGetTransactionManager function. For information about MS DTC, see Microsoft Distributed Transaction Coordinator documentation.

- 2 Call `SQLDriverConnect` or `SQLConnect` once for each Sybase ASE connection you want to establish.
- 3 Call the `ITransactionDispenser::BeginTransaction` function to begin an MS DTC transaction and to obtain an OLE Transaction object that represents the transaction.
- 4 Call `SQLSetConnectAttr` one or more times for each ODBC connection you want to enlist in the MS DTC transaction. `SQLSetConnectAttr` must be called with an attribute of `SQL_ATTR_ENLIST_IN_DTC` and a `ValuePtr` of the Transaction object (obtained in step 3).
- 5 Call `SQLExecDirect` one or more times for each insert or update SQL statement.
- 6 Call the `ITransaction::Commit` function to commit the MS DTC transaction. The Transaction object is no longer valid.

To perform a series of MS DTC transactions, repeat steps 3 through 6.

To release the reference to the Transaction object, call the `ITransaction::Release` function.

To use an ODBC connection with an MS DTC transaction and then use the same connection with a local ASE Server transaction, call `SQLSetConnectAttr` with a `ValuePtr` of `SQL_DTC_DONE` to unenlist the connection from the transaction.

Note Also, you can call `SQLSetConnectAttr` and `SQLExecDirect` separately for each ASE Server, instead of calling them as suggested in steps 4 and 5.

Programming components deployed in Sybase EAServer, MTS, or COM+

The following procedure describes how to create components that participate in Distributed Transactions in Sybase EAServer, MTS, or COM+.

- ❖ **To program components deployed in Sybase EAServer, MTS or COM+**
 - 1 Call `SQLDriverConnect` once for each Sybase ASE connection you want to establish.
 - 2 Call `SQLExecDirect` once for each insert or update SQL statement.

- 3 Deploy your component to MTS, and configure the transaction attributes as needed.

The transaction coordinator creates a distributed transaction as needed, and the component that uses the ASE ODBC driver automatically enlists in the global transaction. Then, the transaction coordinator will commit or roll back the distributed transaction.

Connection properties for Distributed Transaction support

The following describes the Connection properties:

- Distributed Transaction Protocol (DistributedTransactionProtocol) – To specify the protocol used to support the distributed transaction, either XA Interface standard or MS DTC OLE Native protocol, select the Distributed Transaction Protocol in the ODBC Data Source Dialog, or set the property DistributedTransactionProtocol = *OLE* native protocol in the connection string. The default is *XA*.
- Tightly Coupled Transaction (TightlyCoupledTransaction) – When a distributed transaction using two resource managers points to the same ASE server, you have a "Tightly Coupled Transaction." Under these conditions, if you do not set this property to 1, the Distributed Transaction may fail.

To summarize, if you open two database connections to the same ASE server and then enlist these connections in the same distributed transaction, you must set TightlyCoupledTransaction=1. To set this property select the Tightly Coupled Transaction in the ODBC Data Source dialog box, or pass the property TightlyCoupledTransaction=1 in the connection string.

Warning! Enlistment with SQLSetConnectAttr returns a SQL_ERROR if the connection has already begun a local transaction, either by using SQLSetConnectAttr with the SQL_AUTOCOMMIT_OFF or by executing the BEGIN TRANSACTION statement explicitly using SQLExecDirect.

Using directory services

Directory services allow the ASE ODBC Driver to get connection and other information from a central LDAP server; then, it uses this information to connect to an ASE server. It uses a property called Directory Service URL (DSURL), that indicates which LDAP server to use.

LDAP as a directory service

Lightweight Directory Access Protocol (LDAP) is an industry standard for accessing directory services. Directory services allow components to look up information by a distinguished name (DN) from an LDAP server that stores and manages server, user, and software information that is used throughout the enterprise or over a network.

LDAP defines the communication protocol and the contents of messages exchanged between clients and servers. The LDAP server can store and retrieve information about:

- Adaptive Server, such as IP address, port number, and network protocol
- Security mechanisms and filters
- High-availability companion server name

See the Adaptive Server Enterprise *System Administration Guide* for more information.

The LDAP server can be configured with these access restrictions:

- Anonymous authentication – all data is visible to any user.
- User name and password authentication – Adaptive Server uses the default user name and password from the file.

User name and password authentication properties establish and end a session connection to an LDAP server.

Note The LDAP server can be located on a different platform from the one on which Adaptive Server or the clients are running.

Using directory services

To use directory services, add the following properties to the `ConnectionString`:

```
DSURL=ldap://SYBLDAP:389/dc=sybase,dc=com??one?sybase
Servername=MANGO
```

The URL is an LDAP URL and uses LDAP libraries to resolve the URL.

To support high availability on the LDAP server, the DSURL accepts multiple URLs, separated by a semicolon:

```
DSURL={ldap://SYBLDAP:389/dc=sybase,dc=com??one?sybase
Servername=MANGO};
```

The provider attempts to get the properties from the LDAP servers in the order specified. For example:

```
ldap://hostport/dn[?attrs[?scope[?filter[?userdn?userp
ass]]]]
```

where:

- *hostport* is a host name with an optional portnumber, for example, SYBLDAP1:389.
- *dn* is the search base, for example, dc=sybase,dc=com.
- *attrs* is a comma-separated list of attributes requested from the LDAP server. You must leave it blank. Data Provider requires all attributes.
- *scope* is one of three strings:
 - *base* (the default) searches the base.
 - *one* searches immediate children.
 - *sub* searches the sub-tree.
- *filter* is the search filter. Generally, it is the sybaseServername. You can leave it blank and set the Data Source or Server Name property in the `ConnectionString`.
- *userdn* is the user's distinguished name (dn). If the LDAP server does not support anonymous login, you can set the user's dn here, or you can set the `DSPPrincipal` property in the `ConnectionString`.
- *userpass* is the password. If the LDAP server does not support anonymous login, you can set the password here, or you can set the `DSPassword` property in the `ConnectionString`.

The URL can contain *sybaseServername*, or you can set the property *Server Name* to the service name of the LDAP Sybase server object.

The following properties are useful when using Directory Services:

- *DSURL* – set to LDAP URL. The default is an empty string.
- *Server* – the Service Name of the LDAP Sybase server object. The default is an empty string.
- *DSPincipal* – the user name to log in to the LDAP server if it is not a part of *DSURL* and the LDAP server does not allow anonymous access.
- *DSPassword* or *Directory Service Password* – the password to authenticate on the LDAP server if it is not a part of *DSURL* and the LDAP server does not allow anonymous access.

Enabling directory services

This section describes how to enable directory services on the platform you are using.

❖ To enable directory services on Windows

- 1 Launch the ODBC DataSource Administrator.
- 2 Select the data source that you want to use and choose *Configure*.
- 3 Click the *Connection* tab.
- 4 In the *Directory Service Information* group, provide the complete URL in the *URL* field. You also have the option to provide the user name in the *User ID* field and the LDAP Service Name in the *Service Name* Field, to log on to the LDAP server.

Linux

❖ To enable directory services for Linux

Install the following packages:

- *openldap-2.0* (runtime)
- *openldap-devel-2.0*

The ODBC Driver attempts to load a file named *libldap.so*, but to create a symbolic link with this file, you must install the *openldap-devel* package. The *openldap* runtime package does not create the symbolic link.

If you are linking to the unixODBC Driver Manager:

- 1 Edit the ASE ODBC data source template, *odbc.ini*.
- 2 Reinstall the data source using the unixODBC command line tool:

```
# odbcinst -i -s -f <dsn template file>
```

where *dsn template file* is the complete path to the ASE ODBC data source template file.

If you are directly linking to the ASE ODBC Driver, modify the *odbc.ini* file.

Following is an example of the *odbc.ini* data source template file:

```
[sampledsn] Description=Sybase
ODBC Data Source UserID=sa
Password= Driver=Adaptive
Server Enterprise Server=sampleserver
Port=4100
Database=pubs2
UseCursor=1
DSURL=ldap://SYBLDAP1:389/dc=sybase,dc=com??one?sybase
Servername=MANGO
```

Mac OS X

❖ To enable directory services for Mac OS X

- 1 Launch the iODBC Administrator from Applications | Utilities.
- 2 Select the data source you want to use and add two new keyword value pairs:

```
DSURL=ldap://SYBLDAP1:389/dc=sybase,dc=com??one?syb
aseServername=MANGO
```

Bookmark and bulk support

Sybase supports bookmarks and SQL bulk operations for the ODBC Driver.

Bulk insertions that use SQLBulkOperations with the option of SQL_ADD and cursor positioned updates & deletes using SQLSetPos (SQL_UPDATE, SQL_DELETE, SQL_POSITION). For instructions on using SQL_ADD and SQLSetPos, refer to the Microsoft Developer Network library, ODBC Programmer's Reference at <http://msdn.microsoft.com/en-us/library/ms714177.aspx>.

DSN Migration tool

The ODBC DSN Migration tool can help you migrate from the Data Direct ODBC driver to the ODBC Driver by Sybase.

Using the Migration tool

The dsnmigrate tool uses switches to control which DSNs are migrated. You need to enter the following from the command line:

```
dsnmigrate.exe [/?|/help] [l|/ul|/sl] [/a|/ua|/sa]
[ [/dsn|/udsn|/sdsn]=dsn] [/suffix=suffix]
```

All DSNs that are converted are renamed to "<dsn>-backup" before the conversion is completed. When the new Sybase DSNs are created and the conversion is completed, the name is changed to "<dsn>," which will allow existing applications to continue to run without any modifications.

Conversion switches

The following table lists and describes the switches used in the conversion.

Table 3-1: Conversion switches

Switches	Description of results
/?,/h,/help	Displays this message. This message is also displayed if dsnmigrate is called with no command line arguments.
/l	Displays a list of all Sybase Data Direct user and system DSNs.
/ul	Displays a list of all Sybase Data Direct user DSNs.

Switches	Description of results
/sl	Displays a list of all Sybase Data Direct system DSNs.
/a	Converts all Sybase Data Direct user and system DSNs.
/ua	Converts all Sybase Data Direct user DSNs.
/sa	Converts all Sybase Data Direct system DSNs.
/dsn	Converts specific Sybase Data Direct user or system DSNs.
/udsn	Converts specific Sybase Data Direct user DSNs.
/sdsn	Converts specific Sybase Data Direct system DSNs.
dsn	The name of the DSN to be converted.
/suffix	An optional switch that changes the way DSNs are named. If this switch is used, the original DSN is retained and the new DSN is named "<dsn>-<suffix>."
suffix	The suffix that is used to name the new DSN.

Using password encryption

By default, the ASE ODBC Driver sends plain text passwords over the network to ASE for authentication. However, ODBC Driver also supports symmetrical and asymmetrical password encryption, and you can use this feature to change the default behavior and encrypt your password before they are sent over the network.

The symmetrical encryption mechanism uses the same key to encrypt and decrypt the password whereas an asymmetrical encryption mechanism uses one key (the public key) to encrypt the password and another key (the private key) to decrypt the password. Because the private key is not shared across the network, the asymmetrical encryption is considered more secure than symmetrical encryption. When password encryption is enabled, and the server supports asymmetric encryption, this format is used instead of symmetric encryption.

To enable password encryption, you must set the EncryptPassword connection property, which specifies whether the password is transmitted in encrypted format. When password encryption is enabled, the password is sent over the wire only after a login is negotiated; the password is first encrypted and then sent. The EncryptPassword values are:

- 0 – Use plain text password. This is the the default value.
- 1 – Use encrypted password. If it is not supported, return an error message.
- 2 – Use encrypted password. If it is not supported, use plain text password.

Note To use the password encryption feature, you require a server that supports extended password encryption, such as ASE 15.0.2.

Note When using asymmetrical encryption, you may experience a slight delay in login time due to the additional processing time required for asymmetrical encryption.

Windows

❖ To encrypt passwords on Windows

- 1 Launch the ODBC DataSource Administrator.
- 2 Select the data source you want to use and choose Configure.
- 3 Click the Advanced tab.
- 4 Select EncryptPassword.

You can use the EncryptPassword connection property in a call to `SQLDriverConnect`.

Note You can only enable or disable password encryption (which corresponds to EncryptPassword value of 1 and 0, respectively) from the user interface. You can set EncryptPassword to 2 from a connection string.

Linux

To link to the unixODBC Driver Manager, edit the data source template and reinstall the data source using the unixODBC command line tool:

```
# odbcinst -i -s -f dsn template file
```

where *dsn template file* is the complete path to the ASE ODBC Data source template file.

Note If you are directly linking to the ASE ODBC Driver, modify the *odbc.ini* file.

The following is an example of an *odbc.ini* data source template file:

```
[sampledsn] Description=Sybase ODBC Data Source
UserID=sa
Password=
Driver=Adaptive Server Enterprise
Server=sampleserver Port=4100
Database=pubs2
UseCursor=1
EncryptPassword=1
```

Mac OS X

❖ To encrypt passwords on Mac OS X

- 1 Launch the iODBC Administrator from Applications | Utilities.
- 2 Select the data source you want to use and add a new keyword value pair:

```
EncryptPassword=1
```

Password expiration handling

Every company has a specific set of password policies for its database system. Depending on the policies, the password expires at a specific date and time. Unless the password is reset, the ASE drivers connected to a database throw password expired errors and suggest that the user change the password using `isql`. This feature allows users to change their expired passwords using the ASE ODBC Driver.

Changing the password through the connection string properties

To change password, you have to set two connection string properties:

- `oldpassword` – the current password. If `oldpassword` contains a value that is not null or an empty string, the current password is changed to the value contained in `pwd`.
- `pwd` – contains the value of the new password entered by the user. If `oldpassword` does not exist or is null, `pwd` contains the value of the current password.

Changing the password through a dialog box

A change password dialog is activated when “`SQLDriverConnect` with `SQL_DRIVER_PROMPT`” is set to `True`. In this dialog, the user is prompted for the current password and the new password that will replace it.

Using SSL

Secure Sockets Layer (SSL) is an industry standard for sending wire- or socket-level encrypted data over client-to-server and server-to-server connections. Before the SSL connection is established, the server and the client negotiate and agree upon a secure encrypted session. This is called the “SSL handshake.”

Note Additional overhead is required to establish a secure session, because data increases in size when it is encrypted; it also requires additional computation to encrypt or decrypt information. Under normal circumstances, the additional I/O accrued during the SSL handshake can make user login 10 to 20 times slower.

SSL handshake

When a client application requests a connection, the SSL-enabled server presents its certificate to prove its identity before data is transmitted. Essentially, the SSL handshake consists of the following steps:

- 1 The client sends a connection request to the server. The request includes the SSL (or Transport Layer Security, TLS) options that the client supports.
- 2 The server returns its certificate and a list of supported CipherSuites, which includes SSL/TLS support options, the algorithms used for key exchange, and digital signatures. CipherSuites are preferential lists of key-exchange algorithms, hashing methods, and encryption methods used by the SSL protocol.

- 3 A secure, encrypted session is established when both client and server have agreed upon a CipherSuite.

CipherSuites

During the SSL handshake, the client and server negotiate a common security protocol through a CipherSuite.

By default, the strongest CipherSuite supported by both the client and the server is the CipherSuite used for the SSL-based session. Server connection attributes are specified in the connection string or through directory services such as LDAP.

The ASE ODBC Driver and Adaptive Server support the CipherSuites that are available with the SSL Plus library API and the cryptographic engine, Security Builder, both from Certicom Corp.

Note The following list of CipherSuites conform to the Transport Layer Security (TLS) specification. TLS is an enhanced version of SSL 3.0, and is an alias for the SSL version 3.0 CipherSuites.

Following is a list of the CipherSuites, ordered from the strongest to the weakest, supported in ASE ODBC Driver:

- TLS_RSA_WITH_3DES_EDE_CBC_SHA
- TLS_RSA_WITH_RC4_128_SHA
- TLS_RSA_WITH_RC4_128_MD5
- TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA
- TLS_DHE_DSS_WITH_RC4_128_SHA
- TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA
- TLS_RSA_WITH_DES_CBC_SHA
- TLS_DHE_DSS_WITH_DES_CBC_SHA
- TLS_DHE_RSA_WITH_DES_CBC_SHA
- TLS_RSA_EXPORT1024_WITH_DES_CBC_SHA
- TLS_RSA_EXPORT1024_WITH_RC4_56_SHA
- TLS_DHE_DSS_EXPORT1024_WITH_RC4_56_SHA
- TLS_DHE_DSS_EXPORT1024_WITH_DES_CBC_SHA
- TLS_RSA_EXPORT_WITH_RC4_40_MD5
- TLS_RSA_EXPORT_WITH_DES40_CBC_SHA

- TLS_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA
- TLS_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA

For more specific information about the SSL handshake and the SSL/TLS protocol, see the Internet Engineering Task Force Web site at <http://www.ietf.org>.

For a complete description of CipherSuites, go to the IETF organization Web site at <http://www.ietf.org/rfc/rfc2246.txt>.

SSL security levels in ASE ODBC Driver

In ASE ODBC Driver, SSL provides the following levels of security:

- When the SSL session is established, user name and password are transmitted over a secure, encrypted connection.
- When establishing a connection to an SSL-enabled server, the server authenticates itself—proves that it is the server you intended to contact—and an encrypted SSL session begins before any data is transmitted.
- A comparison of the server certificate's digital signature can determine if any information received from the server was modified in transit.

Validating the server by its certificate

Any ASE ODBC Driver client connection to an SSL-enabled server requires that the server have a certificate file, which consists of the server's certificate and an encrypted private key. The certificate must also be digitally signed by a signing/certification authority (CA). ASE ODBC Driver client applications establish a socket connection to Adaptive Server similar to the way that existing client connections are established. Before any user data is transmitted, an SSL handshake occurs on the socket when the network transport-level connect call completes on the client side and the accept call completes on the server side.

To make a successful connection to an SSL-enabled server, the following must occur:

- 1 The SSL-enabled server must present its certificate when the client application makes a connection request.
- 2 The client application must recognize the CA that signed the certificate. A list of all "trusted" CAs is in the "trusted roots file."

The trusted roots file

The list of known and trusted CAs is maintained in the trusted roots file. The trusted roots file is similar in format to a certificate file, except that it contains certificates for CAs known to the entity (such as client applications, servers, network resources, and so on). The System Security Officer adds and deletes trusted CAs using a standard ASCII-text editor.

The application program specifies the location of the trusted roots file using the `TrustedFile=trusted file path` property in the `ConnectionString`. A trusted roots file with the most widely used CAs (Thawte, Entrust, Baltimore, VeriSign, and RSA) is installed in a file located at `$SYBASE/config/trusted.txt`.

For more information about certificates, see the Open Client *Client-Library C Reference Manual*.

Enabling SSL connections

To enable SSL for ASE ODBC Driver, add `Encryption=ssl` and `TrustedFile=<filename>` (where *filename* is the path to the *trusted roots* file) to the `ConnectionString`. The ASE ODBC Driver then negotiates an SSL connection with the ASE server.

Note ASE must be configured to use SSL. For more information on SSL, see the Adaptive Server Enterprise *System Administration Guide*.

Windows

Using a connection string

Before you enable SSL, you must set the `TrustedFile` property in the connection string to the file name of the trusted roots file. The file name should contain the path to the file as well.

Using DSN

❖ **To enable SSL connections**

- 1 Set the `Encryption` property in the connection string to `ssl`.
- 2 Launch the ODBC DataSource Administrator.
- 3 Select the data source name (DSN) you would like to use and choose **Configure**.
- 4 Click the **Connection** tab.
- 5 Select **UseSSL** in the **Secure Socket Layer Group**.

- 6 Provide the complete path to the trusted roots file in the TrustedFile field.

Linux

❖ To enable SSL connections

- 1 Start the unixODBC Driver Manager `odbcinst` utility.
- 2 Open an existing data source template or create a new one.
- 3 Edit the data source template by adding the following:

```
Encryption=ssl
TrustedFile=<filename>line
```

- 4 Reinstall the data source using the unixODBC command line tool:

```
# odbcinst -i -s -f dsn template file
```

where *dsn template file* is the complete path to the ASE ODBC data source template file.

If you are linking directly to the ASE ODBC Driver, modify the *odbc.ini* file.

Following is an example of the *odbc.ini* data source template file:

```
[sampledsn]
Description=Sybase ODBC Data Source
UserID=sa
Password=
Driver=Adaptive Server Enterprise
Server=sampleserver
Port=4100
Database=pubs2
UseCursor=1
Encryption=ssl
TrustedFile=<SYBASE>/config/trusted.txt
```

Mac OS X

❖ To enable SSL connections on Mac OS X

- 1 Launch the iODBC Administrator from Applications | Utilities.
- 2 Select the data source you want to use and add two new keyword value pair:

```
Encryption=ssl
TrustedFile=<filename>
```

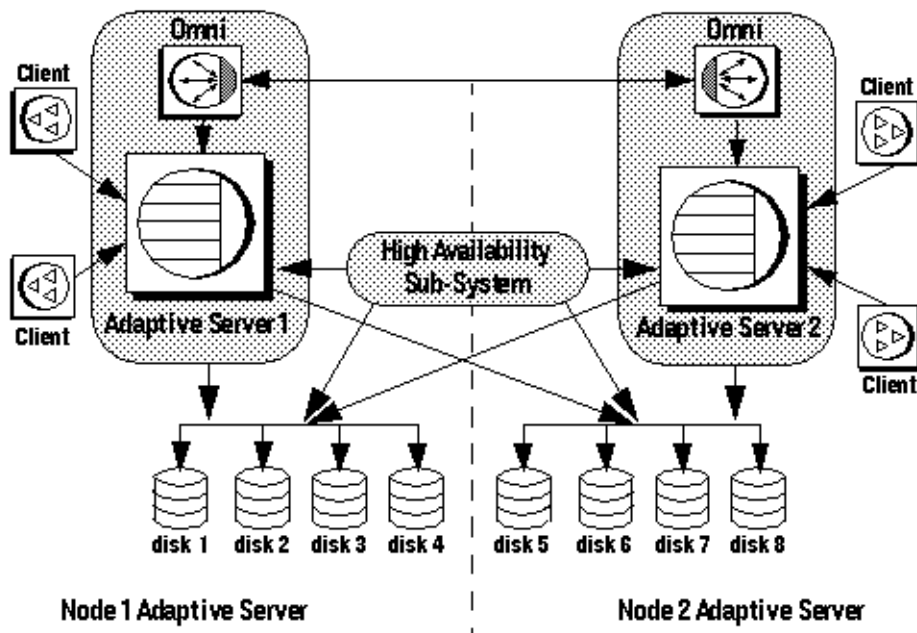
where *<filename>* is the complete path to the trusted roots file.

Using failover in high availability systems

A high availability cluster includes two or more machines that are configured so that if one machine (or application) is interrupted, the second machine assumes the workload of both machines. Each of these machines is called one node of the high availability cluster. A high availability cluster is used in an environment that must always be available, such as, a banking system to which clients must connect continuously, 365 days a year.

The machines in Figure 3-1 are configured so that each machine can read the other machine's disks, although not at the same time. (All of the disks that are failed-over should be shared disks).

Figure 3-1: High availability cluster using failover



For example, if Adaptive Server 1 is the primary companion server, and it crashes, Adaptive Server 2, as the secondary companion server, reads its disks (disks 1 - 4) and manages any databases on them until Adaptive Server 1 can be rebooted. Any clients connected to Adaptive Server 1 are automatically connected to Adaptive Server 2.

Failover allows Adaptive Server to work in a high availability cluster in active-active or active-passive configuration.

During failover, clients connected to the primary companion using the failover property automatically reestablish their network connections to the secondary companion. Failover can be enabled by setting the connection property `HASession` to "1" (default value is "0"). If this property is not set, the session failover does not occur, even if the server is configured for failover. You also must set `SecondaryServer` (the IP address or the machine name of the secondary ASE server) and `SecondaryPort` (the port number of the secondary ASE server) properties. See the ASE book, *Using Sybase Failover in a High Availability System*, for information, about configuring your system for high availability.

When the ASE ODBC driver detects a connection failure with the primary ASE server, it first tries to reconnect to the primary. If it cannot reconnect, it assumes that a failover has occurred. Then, it automatically tries to connect to the secondary ASE server using the connection properties set in `SecondaryServer`, and `SecondaryPort`.

Confirming a successful failover

If a connection to the secondary server is established, the ASE ODBC Driver returns `SQL_ERROR` for the function return code. To confirm a successful failover, you should further examine the `SQLState` and `NativeError` for values of "08S01" and "30130" respectively. The error message returned on such failover is:

```
"Connection to Sybase server has been lost
, you have been successfully connected
to the next available HA server.
All active transactions have been rolled
back."
```

You can access these values by calling `SQLGetDiagRec` on the `StatementHandle`. Then, the client must reapply the failed transaction with the new connection. If failover happens while a transaction is open, only changes that were committed to the database before failover are retained.

Verifying an unsuccessful failover

If the connection to the secondary server is *not* established, the ASE ODBC Driver returns SQL_ERROR for the function return code. To confirm that failover did *not* occur, you should further examine the SQLState and NativeError for values of "08S01" and "30131." The error message returned on an unsuccessful failover is:

```
"Connection to Sybase server has been lost,
connection to the next available HA server
also failed. All active transactions
have been rolled back".
```

You can access these values by calling SQLGetDiagRec on the StatementHandle.

The following shows how to code for a failover:

```
/* Declare required variables */
....
/* Open Database connection */
....
/* Perform a transaction */
...
/* Check return code and handle failover */
if( retcode == SQL_ERROR )
{
    retcode = SQLGetDiagRec(stmt, 1,
        sqlstate,&NativeError, errmsg,100, NULL );
    if(retcode == SQL_SUCCESS ||
        retcode == SQL_SUCCESS_WITH_INFO)
    {
        if(NativeError == 30130 )
        {
            /* Successful failover retry transaction*/
            ...
        }
        else if (NativeError == 30131)
        {
            /* Failover failed. Return error */
            ...
        }
    }
}
```

Windows

The following describes how to use failover for Windows.

❖ **To use failover on Windows**

- 1 Launch the ODBC DataSource Administrator.
- 2 Select the data source you want to use and choose Configure.
- 3 Click the Connection tab.
- 4 Select Enable High Availability in the High Availability Information Group.
- 5 Provide the failover server name in the Server Name field.
- 6 Provide the failover server port in the Server Port field.

Linux

The following describes how to use failover on Linux.

If you are linking to the unixODBC Driver Manager, edit the data source template and reinstall the data source using the unixODBC command line tool:

```
# odbcinst -i -s -f dsn template file
```

where *dsn template file* is the complete path to the ASE ODBC data source template file.

Note If you are directly linking to the ASE ODBC Driver, modify the *odbc.ini* file.

Following is an example of the *odbc.ini* data source template file:

```
[sampledsn]
Driver=Adaptive Server Enterprise
Server=sampleserver
Port=4100
UserID=sa
Password=
Database=pubs2
HASession=1
SecondaryHost=failoverserver
SecondaryPort=5000
```


Mac OS X

❖ To use failover on Mac OS X

- 1 Launch the iODBC Administrator from Applications | Utilities.
- 2 Select the data source you want to use and add three new keyword value pairs:

```
HASession=1  
SecondaryHost=failoverserver  
SecondaryPort=5000
```

Enabling Kerberos authentication

Kerberos is an industry standard network authentication system that provides simple login authentication as well as mutual login authentication. It is used for single sign-on across various applications in extremely secure environments. Instead of passing passwords around the network, a Kerberos server holds encrypted versions of the passwords for users as well as available services.

In addition, Kerberos uses encryption to provide confidentiality and data integrity.

Adaptive Server and the ASE ODBC driver provide support for Kerberos connections. The ASE ODBC driver specifically supports MIT, CyberSafe, and Active Directory (Key Distribution Centers, called KDCs).

Process overview

The Kerberos authentication process works as follows:

- 1 A client application requests a “ticket” from the Kerberos server to access a specific service.
- 2 The Kerberos server returns the ticket, which contains two packets, to the client: The first packet is encrypted using the user password. The second packet is encrypted using the service password. Inside each of these packets is a “session key.”
- 3 The client decrypts the user packet to get the session key.

- 4 The client creates a new authentication packet and encrypts it using the session key.
- 5 The client sends the authentication packet and the service packet to the service.
- 6 The service decrypts the service packet to get the session key and decrypts the authentication packet to get the user information.
- 7 The service compares the user information from the authentication packet with the user information that was also contained in the service packet. If the two match, the user has been authenticated.
- 8 The service creates a confirmation packet that contains service specific information, as well as validation data contained in the authentication packet.
- 9 The service encrypts this data with the session key and returns it to the client.
- 10 The client uses the session key obtained from the user packet it received from Kerberos to decrypt the packet and validates that the service is what it claims to be.

In this way, the user and the service are mutually authenticated. All future communication between the client and the service (in this case, the Adaptive Server database server) will be encrypted using the session key. This successfully protects all data sent between the service and client from unwanted viewers.

Requirements

To use Kerberos as an authentication system, you must configure Adaptive Server Enterprise to delegate authentication to Kerberos. See the Adaptive Server Enterprise *System Administration Guide* for more information.

If Adaptive Server has been configured to use Kerberos, any client that interacts with Adaptive Server must have a Kerberos client library installed. This varies for operating system vendors, as follows:

- On Windows, the Active Directory client library comes installed with the operating system.
- CyberSafe and MIT client libraries are available for Windows and Linux.

For additional information, refer to vendor documentation.

Enabling Kerberos authentication

To enable Kerberos authentication for the ASE ODBC driver, add the following connection properties:

```
AuthenticationClient=<one of 'mitkerberos'
or 'cybersafekkerberos' or 'activedirectory'>
and ServerPrincipal=<ASE server name
```

where *<ASE server name>* is the logical name or the principal as configured in the Key Distribution Center (KDC). The ASE ODBC driver will use this information to negotiate Kerberos authentication with the configured KDC and ASE server.

The Kerberos client libraries are compatible across various KDCs. For example, on Linux you can set `AuthenticationClient` equal to `mitkerberos`, even if your KDC is a Microsoft Active Directory.

If you want the Kerberos client to look for the Ticket Granting Ticket (TGT) in another cache, you might want to specify the `userprincipal` property.

If you use `SQLDriverConnect` with the `SQL_DRIVER_NOPROMPT`, `ConnectionString` appears similar to the following:

```
"Driver=Adaptive Server Enterprise;UID=sa;
PWD='';Server=sampleserver;
Port=4100;Database=pubs2;
AuthenticationClient=mitkerberos;
ServerPrincipal=MANGO;"
```

Windows

❖ To enable Kerberos for Login Authentication on Windows

- 1 Start the Microsoft Windows ODBC Data Source administrator.
- 2 Select the Sybase Adaptive Server Enterprise ODBC Driver.
- 3 Select the User DSN/ System DSN tab and click the data source that you would like to modify, or choose Add New Data Source.
- 4 On the Security tab, select Use Active Directory under the Kerberos Authentication Client.
- 5 Enter the name of the server principal in the Server Principal edit box. This name should match the name of the ASE server configured in the KDC.

Linux

❖ To enable Kerberos for Login Authentication on Linux

If you are linking to the UNIX ODBC Driver Manager:

- 1 Open an existing data source, or create a new data source template.
- 2 Add the following to the data source template:

```
Authentication= mitkerberos
(or cybersafekkerberos) ServerPrincipal=<MANGO>
to enable Kerberos Login Authentication.
```

where: <MANGO> is the name of the principal server used to authenticate sign-ons.

- 3 Reinstall the data source using the `odbcinst` utility at the command line:

```
odbcinst -i-s -f ${datasourcetemplatefile}
```

If you are linking directly to the ASE ODBC Driver, modify the `odbc.ini` file directly.

Following is an example of how the `odbc.ini` data source template file should look after you modify it:

```
[sampledsn]
Description=Sybase ODBC Data Source
UserID=sa
Password=
Driver=Adaptive Server Enterprise
Server=sampleserver
Port=4100
Database=pubs2
UseCursor=1
AuthenticationClient=mitkerberos
ServerPrincipal=MANGO
```

Obtaining an initial ticket from the Key Distribution Center

To use Kerberos authentication, you must generate an initial ticket called Ticket Granted Ticket (TGT) from the Key Distribution Center. The procedure to obtain this ticket depends on the Kerberos libraries being used. For more information, refer to the vendor documentation.

❖ To generate TGTs for the MIT Kerberos client library

- 1 Start the `kinit` utility at the command line:

```
% kinit
```

- 2 Enter the kinit user name, such as *your_name@YOUR.REALM*.
- 3 Enter the password for *your_name@YOUR.REALM*, such as *my_password*. When you enter your password, the kinit utility submits a request to the Authentication Server for a Ticket Granting Ticket (TGT).

The password is used to compute a key, which in turn is used to decrypt part of the response. The response contains the confirmation of the request, as well as the session key. If you entered your password correctly, you now have a TGT.

- 4 Verify that you have a TGT by entering the following at the command line:

```
% klist
```

The results of the klist command should be:

```
Ticket cache: /var/tmp/krb5cc_1234
Default principal: your_name@YOUR.REALM
Valid starting      Expires      Service principal
24-Jul-95 12:58:02  24-Jul-95 20:58:15  krbtgt/YOUR.REALM@YOUR.REALM
```

Explanation of results

Ticket cache The ticket cache field tells you which file contains your credentials cache.

Default principal The default principal is the login of the person who owns the TGT (in this case, you).

Valid starting/Expires/Service principal The remainder of the output is a list of your existing tickets. Because this is the first ticket you have requested, there is only one ticket listed. The service principal (*krbtgt/YOUR.REALM@YOUR.REALM*) shows that this ticket is a TGT. Note that this ticket is good for approximately 8 hours.

Index

A

- advanced sample 23
- allocating 8
- authentication 69

B

- bound parameters 13

C

- certificate 62
- CipherSuites 61
- connecting to a data source 9
- connection
 - establishing 10
 - how parameters work 32
 - introduction 31
 - setting attributes 12
 - strings 32
 - table of parameters 40
 - using parameters 40
- connection functions 9
- connection handle 7
- conventions ix
- cursor characteristics 17
- cursor sample 18
- cursors
 - choosing characteristics 17
 - updating and deleting rows 18

D

- data
 - retrieving 18
- data source

- connecting to 9
 - connecting with 40
 - template 36
- datatype mapping 26
- datatypes
 - computed columns 28
 - large identifiers 29
- descriptor handle 7
- directly executing SQL statements 13
- directory services 52
 - using 53
- Distributed Transaction Manager (DTC) 49
- DSURL 53

E

- EncryptPassword 57
- environment handle 7
- error handling 25
- establishing connections 10
- executing
 - prepared statements 15
 - SQL statements 12
 - SQL statements directly 13
 - SQL statements with bound parameters 13

F

- failover
 - on Linux 68
 - on Mac OS X 69
 - on Windows 67
 - using in high availability systems 65

H

- handles 6

Index

- allocating 8
- handling errors 25
- handshake 60
- help xi
- high availability systems
 - using failover in 65

K

- Kerberos 69
 - Linux 72
 - process overview 69
 - requirements 70
 - Windows 71
- kinit utility 72

L

- LDAP 52
- Linux
 - failover on 68
 - Kerberos 72

M

- Mac OS X
 - failover on 69

N

- network authentication 69

O

- ODBC
 - backward compatibility 3
 - conformance, conformance 2
 - driver manager 3
 - introduction 2
- odbc.ini file 36

P

- password encryption 57
- prepared statements 15
- process overview
 - Kerberos 69

R

- related documents vii
- requirements
 - Kerberos 70
- result sets 17
- retrieving data 18
- return codes 25

S

- samples
 - advanced 23
 - cursor 18
 - simple 18
- Secure Sockets Layer (SSL)
 - enabling connections 63
 - in ASE ODBC Driver 62
 - using 60
 - validation 62
- setting connections attributes 12
- simple sample 18
- SQL statements
 - executing 12
 - executing directly 13
 - executing prepared statements 15
 - executing with bound parameters 13
- SSL see Secure Sockets Layer 60
- statement handle 7
- stored procedures
 - calling 23

T

- threads 12
- trusted roots file 63

U

updating and deleting rows through a cursor 18

V

validation 62

W

Windows

 failover on 67

 Kerberos 71

