User’s Guide

Adaptive Server® Enterprise ODBC Driver by Sybase®

12.5.1

[Windows, Linux, Mac OS X]

DOCUMENT ID: DC20116-01-1251-03

LAST REVISED: September 2004

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

Contents

About This Book

Audience

This document is intended for application developers who need access to data from Adaptive Server® Enterprise (ASE) on Windows, Linux and Mac OS X 10.3 platforms, using the 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 on installing the Software Developer’s Kit, see the Software Developer’s Kit and Open Server 12.5.1 *Installation Guide*.

For information on known problems and recent updates, see the Software Developer’s Kit and Open Server 12.5.1 *Release Bulletin*.

For information on installing the Adaptive Server Enterprise on Windows, see the Adaptive Server Enterprise 12.5.1 for Windows *Installation Guide*.

For information on installing the Adaptive Server Enterprise on Linux, see the Adaptive Server Enterprise 12.5.1 for Linux *Installation Guide*.

For information on installing the Adaptive Server Enterprise on Mac OS X 10.3, see the Adaptive Server Enterprise 12.5.1 for Mac OS X 10.3 *Quick Installation Guide*.

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

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

For information on known problems and recent updates to Adaptive Server Enterprise on Mac OS X 10.3, see the Adaptive Server Enterprise Version 12.5.2 for Mac OS X *Release Bulletin*.

Other sources of information

Use the Sybase® Getting Started CD, the Sybase Technical Library CD, and the Technical Library Product Manuals Web site to learn more about your product:

- The Getting Started CD contains release bulletins and installation guides in PDF format, and might also contain other documents or updated information not included on the Technical Library CD. It is included with your software. To read or print documents on the Getting Started CD, you need Adobe Acrobat Reader (downloadable at no charge from the Adobe Web site, using a link provided on the CD).
- The Technical Library CD contains product manuals and is included with your software. The DynaText reader (included on the Technical Library CD) allows you to access technical information about your product in an easy-to-use format.

  Refer to the *Technical Library Installation Guide* in your documentation package for instructions on installing and starting the Technical Library.
- The Technical Library Product Manuals Web site is an HTML version of the Technical Library CD that you can access using a standard Web browser. In addition to product manuals, you will find links to EBFs/Updates, Technical Documents, Case Management, Solved Cases, newsgroups, and the Sybase Developer Network.

  To access the Technical Library 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.

❖ To find the latest information on product certifications

1. Point your Web browser to Technical Documents at http://www.sybase.com/support/techdocs/.
2. Select Products from the navigation bar on the left.
3. Select a product name from the product list and click Go.
4. Select the Certification Report filter, specify a time frame, and click Go.
5. Click a Certification Report title to display the report.

❖ To create 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

❖ **To find 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. Enter user name and password information, if prompted (for existing Web accounts) or create a new account (a free service).
3. Select a product.
4. Specify a time frame and click Go.
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.

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.

CHAPTER 1

Introduction to ODBC Programming

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

It covers the following topics:

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 from hereon as ASE ODBC Driver) but is not an 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 UNIX and Macintosh.

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 iOBDC 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 (NT 4.0, 2000, XP, and 2003)
- Linux 32-bit (x86 architecture)
- Mac OS X

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 *Microsoft ODBC Programmer's Reference*.

Features supported by ASE ODBC Driver

The ASE ODBC Driver meets Level 2 conformance with the following exceptions:

- **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, and describing dynamic parameters by calling SQLDescribeParam.

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. Then, 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 an ODBC Driver Manager is installed when you install the operating system. Also there are multiple open source and commercial implementations of ODBC Driver Manager 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 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.

❖ 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 `-L`*dir* flag to the linker,

  where:

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

Mac OS X

❖ To link an ODBC application against the iODBC Driver manager

The iODBC Driver Manager includes a dynamic library named *libiodbc.dylib*, typically located in the */usr/lib* directory.

- 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 `-L`*dir* flag to the linker,

where:

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

Building applications not using an ODBC Driver Manager

You cannot build your applications directly against the ASE ODBC Driver on Windows and Mac OS X platforms. You need to build your applications against an ODBC Driver Manager on these platforms.

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 *$SYBASE/DataAccess/ODBC/lib* directory, where *$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, where the *<dir to ASE ODBC Driver>* is usually the *$SYBASE/DataAccess/ODBC/lib* directory.
2. When deploying your application, verify that the directory containing the ASE ODBC Driver shared library (*$SYBASE/DataAccess/ODBC/lib*, where *$SYBASE* is the Sybase installation root directory) 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.

Each directory and sample contains a *README* file that contains instructions on building and running the 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`
Statement	`SQLHSTMT`
Descriptor	`SQLHDESC`

The following handles are used in essentially 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 illustrates how to allocate 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 illustrates how to allocate 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 illustrates how to allocate 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 an SQL statement or the columns of a result set, as seen by the application or driver (also known as the implementation). 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.

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, 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 and Mac platforms, 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*.

In general, the examples in this chapter use SQLConnect.

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_UINTEGER );
```

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 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_UINTEGER );
```

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 so 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.

You set the required ODBC cursor characteristics by calling 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 sufficient 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. The default value for SQL_ATTR_MAX_LENGTH 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. For the complete code, refer to the *cursor.cpp* sample.

```
/* 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) ;
```

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
SQL_SUCCESS	No error.
SQL_SUCCESS_WITH_INFO	The function completed, but a call to SQLGetDiagRec 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.
SQL_INVALID_HANDLE	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.
SQL_NO_DATA	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.
SQL_NEED_DATA	Data is needed for a parameter. This is an advanced feature described in the ODBC Software Development Kit documentation under SQLParamData and SQLPutData.

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 fragment uses 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 type	ODBC bind type
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
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_LONGVARBINARY	SQL_C_BINARY
int[eger]	SQL_INTEGER	SQL_C_LONG

ASE datatype	ODBC SQL type	ODBC bind type
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
univarchar	SQL_WVARCHAR	SQL_C_CHAR
varbinary	SQL_VARBINARY	SQL_C_BINARY
varchar	SQL_VARCHAR	SQL_C_CHAR

Special instructions for unichar and varchar

When you use the ASE datatypes unichar and univarchar, and bind either of them to SQL_C_CHAR, in Linux and Mac, 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. For Mac OS X, they are installed by default.

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.

CHAPTER 2 Connecting to a Database

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

It covers the following topics:

Introduction to connections

Any client application that uses Adaptive Server Enterprise must establish a connection to that 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 username to connect to the server.

[password] is the password for the username. If the value is null, leave the parameter empty.

Linux and Mac OS X

❖ **To install the stored procedures on a target Adaptive Server on Linux and Mac OS X**

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 username to connect to the server.

[password] is the password for the username. 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 assembled into connection strings, in which a semicolon separates each connection parameter. Connection parameters are passed to the ASE ODBC driver as a connection string and are separated by semicolons:

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

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 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 33.

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.

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 actual path to the ASE ODBC Driver installation
- *driver library name* is the actual 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 for the ASE ODBC Driver 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 typically saved in the */etc/odbc.ini* file. If the same data source is defined in both the 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 using 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 user-visible, the *odbcinst.ini* and the *odbc.ini* files are in the *$HOME/Library/ODBC* directory.

At run-time, 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 typically 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 apart 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

Property names	Description	Required	Default value
Server	The name or the IP address of the ASE server.	Yes	Empty
Port	The port number of ASE server.	Yes	Empty
Database	The database to which you want to connect.	No	Empty
UseCursor	Specifies whether cursors are to be used by the driver. 0 indicates do not use cursors and 1 indicates use cursors.	No	0
ApplicationName	The name to be used by ASE to identify the client application.	No	Empty
PacketSize	The number of bytes per network packet transferred between ASE and the client.	No	512
CharSet	The designated character set. The specified character set must be installed on the ASE server.	No	Empty
Language	The language in which ASE returns error messages.	No	Empty – ASE uses English by default
Encryption	The designated encryption. Possible values: ssl.	No	Empty
TrustedFile	If encryption is set to ssl, this property should be set to the path to the Trusted File.	No	Empty
DSURL	The URL to the LDAP server.	No	Empty

Property names	Description	Required	Default value
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
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
DynamicPrepare	When set to 1 the driver sends SQLPrepare calls to ASE to compile/prepare. This can boost performance if you reuse the same query over and over again.	No	0
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

Property names	Description	Required	Default value
QuotedIdentifier	Specifies if ASE treats character strings enclosed in double quotes as identifiers. 0 indicates do not enable quoted identifiers, 1 indicates enable quoted identifiers.	No	0
HASession	Specifies if high availability is enabled. 0 indicates high availability disabled, 1 high availability enabled.	No	0
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
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
EncryptedPassword	Specifies if password encryption is enabled. 0 indicates password encryption is disabled, 1 indicates password encryption is enabled.	No	0
BufferPoolSize	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
CRC	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. Set this property to 0 if you want the driver to return only the last update count.	No	1
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
TextSize	The maximum size of binary or text data that will be sent over the wire.	No	Empty. ASE default is 32K.
AnsiNull	Strict ODBC compliance where you cannot use “= NULL.” Instead, you must use “IsNull.”	No	1

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 make use of this feature and manage the job of starting transactions. 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

CHAPTER 3

ASE Advanced Features

This chapter describes the advanced ASE features you can use with the ASE ODBC Driver. It covers the following topics:

Directory services for Windows

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. Directory services is supported on Windows only.

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 ConnectString:

```
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 with 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.

An example of DSURL follows:

```
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 DSPrincipal 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.
- DSPrincipal — The user name to log on 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 you would want to use and choose Configure.

3 Click the Connection tab.

4 Provide the complete URL in the LDAP URL field. You also have the option to provide the user name to log on to the LDAP server, in the LDAP User ID field.

Password encryption

By default, the ASE ODBC Driver sends plain text passwords over the network to ASE for authentication. You can use this feature to change the default behavior and encrypt passwords before they are sent over the network. When `EncryptPassword` is set to 1, the password is not sent over the wire until a login is negotiated; then, the password is encrypted and sent.

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`.

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
```

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, and it requires additional computation to encrypt or decrypt information. Typically, 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.
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. CipherSuites are preferential lists of key-exchange algorithms, hashing methods, and encryption methods used by the SSL protocol.

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 TLS specification. TLS, or Transport Layer Security, is an enhanced version of SSL 3.0, and is an alias for the SSL version 3.0 CipherSuites.

The following lists the CipherSuites, ordered by strength from strongest to weakest, supported in ASE OBDC 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:

- Once 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 OBDC 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 OBDC Driver client applications establish a socket connection to Adaptive Server similarly 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 (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 ConnectString. A trusted roots file with 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 ConnectString. The ASE ODBC Driver then negotiates an SSL connection with the ASE server.

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

```
char ConnectString[BUFSIZ];
```

```
strcpy(ConnectString, "Driver=Adaptive Server Enterprise;");
strcat(ConnectString, "UserID=sa;Password=;");
strcat(ConnectString, "Server=sampleserver;");
strcat(ConnectString, "Port=4100;Database=pubs2;");
strcat(ConnectString, "UseCursor=1;");
strcat(ConnectString, "Encryption=ssl;");
strcat(ConnectString, "TrustedFile=/opt/sybase/config/trusted.txt");
```

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

Windows

❖ To enable SSL connections on Windows

1. Launch the ODBC DataSource Administrator.
2. Select the data source you would like to use and choose Configure.
3. Click the Connection tab.
4. Select UseSSL in the Secure Socket Layer Group.
5. Provide the complete path to the trusted roots file in the TrustedFile field.

Linux

❖ To enable SSL connections 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.

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

The 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.

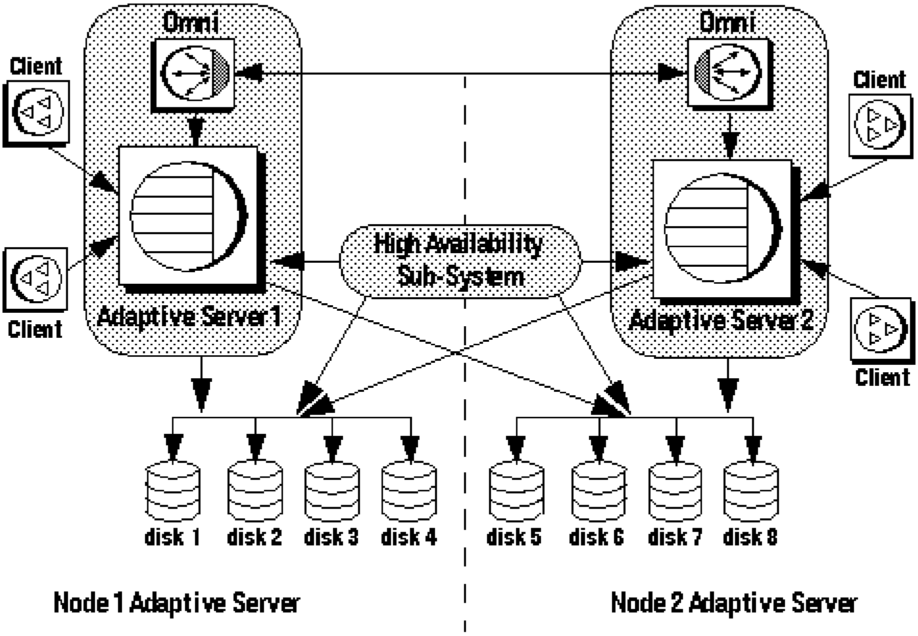
Using failover in a high availability system

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 typically used in an environment that must always be available, for example, 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 the disks that are failed-over should be shared disks).

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.

Figure 3-1: High availability cluster using failover



Failover enables 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 if configured for failover. You also have to 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 a failover has occurred. Then, it automatically tries to connect to the secondary ASE server using the connection properties set in SecondaryServer, SecondaryPort.

If a connection to the secondary server is established, the ASE ODBC Driver returns SQL_ERROR for the function return code. You should further examine the SQLState and NativeError for values of “08S01” and “30130” respectively to confirm a successful failover. 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.

If the connection to the secondary server is *not* established, the ASE ODBC Driver returns SQL_ERROR for the function return code. You should further examine the SQLState and NativeError for values of “08S01” and “30131” to confirm that failover did *not* occur. 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 code snippet 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

❖ **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

❖ **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.

The 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
```

Index