SYBASE®

User's Guide

Adaptive Server® Enterprise OLE DB Provider by Sybase®

15.0

[Microsoft Windows]

DOCUMENT ID: DC00075-01-1500-01

LAST REVISED: September 2005

Contents

About This Book

Audience

This document is intended for application developers who need access to data from Adaptive Server® Enterprise (ASE) on Microsoft Windows platforms using the ASE OLE DB Provider.

How to use this book

The information in this book is organized as follows:

- Chapter 1, “Introduction to ASE OLE DB Provider,” describes OLE DB programming and provides samples.
- Chapter 2, “Connecting to a Database,” describes how to connect to ASE using ASE OLE DB Provider.
- Chapter 3, “ASE Advanced Features,” describes advanced ASE features supported by ASE OLE DB Provider.

Related documents

Software Developer’s Kit and Open Server 12.5.1 *Installation Guide*

Software Developer’s Kit and Open Server 12.5.1 *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. 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.

❖ 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 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 can use IDBCreateSession::CreateSession() to create a session.
- Variables, parameters, and user-supplied words are in italics in syntax and in paragraph text, are printed as follows:

  For example, the statement `int` ***`RowCount`***`;` where *RowCount*; is a variable of type int.
- 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:

```
ICommandText* pICommandText = NULL;
HRESULT hr = pIDBCreateCommand->CreateCommand(NULL,
   IID_ICommandText, (IUnknown**)&pICommandText);
```

```
pIDBCreateCommand->Release();
```

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.

CHAPTER 1

Introduction to ASE OLE DB Provider

This chapter describes how to use the OLE DB interface to get full access to ASE features from an ADO programming environment.

Many applications that use the OLE DB interface do so through the Microsoft ActiveX Data Objects (ADO) programming model, rather than directly. This chapter also describes ADO programming with Adaptive Server. It covers the following topics:

Introduction to OLE DB

OLE DB is a data access model from Microsoft. It uses the Component Object Model (COM) interfaces and, unlike ODBC, does not assume that the data source uses a SQL query processor.

Each OLE DB provider is a dynamic-link library. You need an OLE DB provider for each type of data source you want to access. There are two OLE DB providers you can use to access ASE:

- **Sybase ASE OLE DB Provider** The ASE OLE DB Provider provides access to ASE as an OLE DB data source without the need for ODBC components. The short name for this provider is ASEOLEDB.
- **Microsoft OLE DB provider for ODBC** Microsoft provides an OLE DB provider with a short name of MSDASQL. The MSDASQL provider makes ODBC data sources appear as OLE DB data sources. To do this, it requires the ASE ODBC Driver.

Using the ASE OLE DB Provider brings the following benefits:

- ODBC is not required in your deployment.
- You can get full access to ASE features from OLE DB programming environments. The MSDASQL provider allows OLE DB clients to work with any ODBC driver but does not guarantee that you can use the full range of functionality of each ODBC driver.

Supported platforms

The ASE OLE DB Provider is designed to work with OLE DB 2.5 and later. Supported platforms include Microsoft Windows 2000, 2003, and XP.

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

ADO programming with ASE OLE DB Provider

ActiveX Data Objects (ADO) is a data access object model exposed through an Automation interface, which allows client applications to discover the methods and properties of objects at runtime without any prior knowledge of the object. Automation allows scripting languages like Visual Basic to use a standard data access object model. ADO uses OLE DB to provide access to data on different databases.

Using the ASE OLE DB Provider, you get full access to ASE features from an ADO programming environment.

This section describes how to carry out basic tasks using ADO from Visual Basic. It is not a complete guide to programming using ADO. For information on programming in ADO, see your development tool documentation.

Connecting to a database using the Connection object

This section describes a simple Visual Basic routine that connects to a database.

Sample code

You can try this routine by placing a command button named Command1 on a form, and pasting the routine into its Click event. Run the program and click Command1 to connect and then disconnect.

```
Private Sub cmdTestConnection_Click()
' Declare variables
Dim myConn As New ADODB.Connection
On Error GoTo HandleError
' Establish the connection
myConn.Provider = "ASEOLEDB"
myConn.ConnectionString = _
     "Data Source=MANGO:5000;User ID=sa;Pwd=;"
myConn.Open
MsgBox "Connection succeeded"
myConn.Close
Exit Sub
HandleError:
MsgBox "Connection failed"
Exit Sub
End Sub
```

Notes

The sample carries out the following tasks:

- It declares the variables used in the routine.
- It establishes a connection, using the ASE OLE DB Provider, to the sample database.
- It closes the connection.

When the ASEOLEDB provider is installed, it registers itself. This registration process includes making registry entries in the COM section of the registry, so that ADO can locate the DLL when the ASEOLEDB provider is called. If you change the location of your DLL, you must re-register it using the following steps:

❖ **To register the OLE DB provider**

1. Open a command prompt.
2. Change to the directory where the OLE DB provider is installed.
3. Enter the following command to register the provider:

```
regsvr32 sybdrvoledb.dll
```

Executing statements using the Command object

This section describes a simple routine that sends a simple SQL statement to the database.

Sample code

You can try this routine by placing a command button named Command2 on a form, and pasting the routine into its Click event. Run the program and click Command2 to connect, display a message on the database server window, and then disconnect.

```
Private Sub cmdUpdate_Click()
' Declare variables
Dim myConn As New ADODB.Connection
Dim myCommand As New ADODB.Command
Dim cAffected As Long
' Establish the connection
myConn.Provider = "ASEOLEDB"
myConn.ConnectionString = _
"Data Source = MANGO:5000; User ID=sa;PWD=;"+_
   "Initial Catalog=pubs2;"
myConn.Open
'Execute a command
myCommand.CommandText = _
"INSERT INTO publishers values" +_
"('7777', 'American Books', 'Boston', 'MA')"
Set myCommand.ActiveConnection = myConn
myCommand.Execute cAffected
MsgBox CStr(cAffected) + " rows affected.",
vbInformation
myConn.Close
End Sub
```

Notes

After establishing a connection, the example code creates a Command object, sets its CommandText property to an insert statement, and sets its ActiveConnection property to the current connection. Then, it executes the insert statement and displays the number of rows affected by the update in a message box.

In this example, the insert statement is sent to the database and committed as soon as it is executed.

Querying the database with the Recordset object

The ADO Recordset object represents the result set of a query. You can use it to view data from a database.

Sample code

You can try this routine by placing a command button named cmdQuery on a form and pasting the routine into its Click event. Run the program and click CmdQuery to connect, display a message on the database server window, execute a query and display the first few rows in message boxes, and then disconnect.

```
Private Sub cmdQuery_Click()
' Declare variables
Dim myConn As New ADODB.Connection
Dim myCommand As New ADODB.Command
Dim myRS As New ADODB.Recordset
On Error GoTo ErrorHandler:
' Establish the connection
myConn.Provider = "ASEOLEDB"
myConn.ConnectionString = _
"Data Source = MANGO:5000; User ID=sa;PWD=;" +_
   "Initial Catalog=pubs2;"
myConn.Open
'Execute a query
Set myRS = New Recordset
myRS.CacheSize = 50
myRS.Source = "Select * from customer"
myRS.ActiveConnection = myConn
myRS.LockType = adLockOptimistic
myRS.Open
'Scroll through the first few results
For i = 1 To 5
     MsgBox myRS.Fields("company_name"), vbInformation
     myRS.MoveNext
Next
myRS.Close
myConn.Close
Exit Sub
ErrorHandler:
MsgBox Error(Err)
Exit Sub
End Sub
```

Notes

The Recordset object in this example holds the results from a query on the Customer table. The For loop scrolls through the first several rows and displays the “company_name” value for each row.

This is a simple example of using a cursor from ADO.

Working with the Rowset object

When working with ASE, the ADO Rowset represents a cursor. You can choose the type of cursor by declaring a CursorType property of the Rowset object before you open the Rowset. The choice of cursor type controls the actions you can take on the Rowset and has performance implications.

Cursor types

The set of cursor types supported by ASE is described in the ASE *Transact-SQL User's Guide*.

ADO has its own naming convention for cursor types. Following are the available cursor types, the corresponding cursor type constants, and the ASE types they are equivalent to:

ADO cursor type	ADO constant	ASE type
Forward only	adOpenForwardOnly	No-scroll cursor
Scrollable	adOpenStatic	Scrollable

Sample code

The following code sets the cursor type for an ADO Rowset object:

```
Dim myRS As New ADODB.Rowset myRS.CursorType=_
     adOpenForwardOnly
```

Using transactions

By default, any change you make to the database using ADO is committed as soon as it is executed. This includes explicit updates, as well as the UpdateBatch method on a Recordset. However, the previous section illustrated that you can use the BeginTrans and RollbackTrans or CommitTrans methods on the Connection object to use transactions.

Transaction isolation level is set as a property of the Connection object. The IsolationLevel property can take on one of the following values:

ADO isolation level	Constant	ASE level
Unspecified	adXactUnspecified	Not applicable. Set to 0
Chaos	adXactChaos	Unsupported. Set to 0
Browse	adXactBrowse	0
Read uncommitted	adXactReadUncommitted	0
Cursor stability	adXactCursorStability	1
Read committed	adXactReadCommitted	1
Repeatable read	adXactRepeatableRead	2
Isolated	adXactIsolated	3

ADO isolation level	Constant	ASE level
Serializable	adXactSerializable	3

Supported OLE DB interfaces

The OLE DB API consists of a set of interfaces. The following table describes the support for each interface in the ASE OLE DB Provider.

Table 1-1: Supported OLE DB interfaces

Interface	Purpose	Limitations
IAccessor	Define bindings between client memory and data store values.	DBACCESSOR_PASSBYREF not supported. DBACCESSOR_OPTIMIZED not supported.
IColumnsInfo	Get simple information about the columns of a rowset.	NA
IColumnsRowset	Get information about optional metadata columns in a rowset, and get a rowset of column metadata.	NA
ICommand	Execute SQL commands.	To find properties that could not have been set, it does not support calling. IcommandProperties: GetProperties with DBPROPSET_PROPERTIESINERROR.
ICommandPrepare	Prepare commands.	NA
ICommandProperties	Set Rowset properties for rowsets created by a command. Most commonly used to specify the interfaces the rowset should support.	NA
ICommandText	Set the SQL command text for ICommand.	Only the DBGUID_DEFAULT SQL dialect is supported.

Interface	Purpose	Limitations
ICommandWithParameters	Set or get parameter information for a command.	No support for parameters stored as vectors of scalar values.
IConvertType		NA
IDBCreateCommand	Create commands from a session.	NA
IDBCreateSession	Create a session from a data source object.	NA
IDBInfo	Find information about keywords unique to this provider (that is, find non-standard SQL keywords). Also, find information about literals, special characters used in text matching queries, and other literal information.	NA
IDBInitialize	Initialize data source objects and enumerators.	NA
IDBProperties	Manage properties on a data source object or enumerator.	NA
IDBSchemaRowset	Get information about system tables, in a standard form (a rowset).	NA
IErrorLookup IErrorRecords	Support ActiveX error object.	NA
IGetDataSource	Return an interface pointer to the session's data source object.	NA
IMultipleResults	Retrieve multiple results (rowsets or row counts) from a command.	NA
IOpenRowset	Access a database table by its name, in a non-SQL way.	Opening a table by its name is supported, not by a GUID.
IRowset	Access rowsets.	NA
IRowsetIdentity	Compare row handles.	NA

Interface	Purpose	Limitations
ISequentialStream	Retrieve a blob column.	Supported for reading only. No support for SetData with this interface.
ISessionProperties	Get session property information.	NA
ISourcesRowset	Get a rowset of data source objects and enumerators.	NA
ITableDefinition	Create, drop, and alter tables, with constraints.	NA
ITransaction	Commit or abort transactions.	Not all the flags are supported.

OLE DB programming with ASE OLE DB Provider

This section describes how to carry out basic tasks in OLE DB while using ASE OLE DB Provider.

Connecting to a data source using OLE DB

The following describes how to use OLE DB interfaces to establish a connection to an ASE database.

There are two ways to set up a connection using OLE DB, described as follows:

❖ **To connect using IDBInitialize**

1 Call CoCreateInstance.

2 Pass the clsid obtained from CLSIDFromProgID("ASEOLEDB").

3 Set the connection properties using IDBInitialize.

❖ **To connect using IDataInitialize**

1 Call CoCreateInstance.

2 Pass the clsid obtained from MSDAINITIALIZE.

3 Set the connection properties using IDataInitialize.

Code example

A code example for establishing an OLE DB connection follows:

```
wchar_t* szInitializationString = L"Provider=ASEOLEDB;
   User ID=sa;Password=;Initial Catalog=pubs2;
   Data Source=MANGO:5000;"

IDataInitialize* pIDataInitialize = NULL;
HRESULT hr = CoCreateInstance(
     __uuidof(MSDAINITIALIZE), NULL, CLSCTX_ALL,
     __uuidof(IDataInitialize), (void**)&pIDataInitialize);

IDBInitialize* pIDBInitialize = NULL;
hr = pIDataInitialize->GetDataSource(NULL, CLSCTX_ALL,
     szInitializationString,
     __uuidof(IDBInitialize), (IUnknown**)&pIDBInitialize);
hr = pIDBInitialize->Initialize();

IDBCreateSession* pIDBCreateSession = NULL;
hr = pIDBInitialize->QueryInterface(
     IID_IDBCreateSession, (void**)&pIDBCreateSession);

IDBCreateCommand* pIDBCreateCommand = NULL;
hr = pIDBCreateSession->CreateSession(NULL,
     IID_IDBCreateCommand,
     (IUnknown**)&pIDBCreateCommand);

ICommandText* pICommandText = NULL;
hr = pIDBCreateCommand->CreateCommand(NULL,
     IID_ICommandText, (IUnknown**)&pICommandText);

// use the command object
// ...

pICommandText->Release();

pIDBCreateSession->Release();
pIDBCreateCommand->Release();
pIDBInitialize->Release();
pIDataInitialize->Release();
```

Using threads and connections in OLE DB applications

You can develop multithreaded OLE DB applications for ASE. 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

OLE DB 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. Bound 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 you want to execute repeatedly, this avoids repeated preparation and, as a result, improves performance.

Executing statements directly

The ICommandText::Execute() function prepares and executes a SQL statement. The code samples in this section describe how to execute a statement without parameters. Optionally, the statement can include parameters.

❖ **To execute a statement without parameters**

1 Obtain a Command object from the session:

```
ICommandText* pICommandText;
hr = pIDBCreateCommand->CreateCommand(
     NULL, IID_ICommandText,
     (IUnknown**)&pICommandText);
```

2 Set the SQL statement the command will execute:

```
hr = pICommandText->SetCommandText(
     DBGUID_DBSQL,
     L"DELETE FROM publishers where pub_id = '7777'
");
```

3 Execute the command. The `cRowsAffected` contain the number of rows inserted, deleted, or updated by the command. The `pIRowset` is assigned to the Rowset object created by the command, as shown:

```
DBROWCOUNT cRowsAffected;
IRowset* pIRowset;
hr = pICommandText->Execute(
          NULL, IID_IRowset, NULL,
          &cRowsAffected, (IUnknown**)&pIRowset);
```

Executing statements with bound parameters

The code samples in this section describe how to construct and execute a SQL statement, using bound parameters to set values for statement parameters at runtime.

❖ **To construct and execute a SQL statement**

1 Create a Command object from the session:

```
ICommandText* pICommandText;
hr = pIDBCreateCommand->CreateCommand(
     NULL, IID_ICommandText,
     (IUnknown**)&pICommandText);
```

2 Set the SQL statement you want to execute:

```
hr = pICommandText->SetCommandText(
     DBGUID_DBSQL,
     L"DELETE FROM department WHERE dept_id = ?");
```

3 Create an array to describe the parameters:

```
DB_UPARAMS paramOrdinal[1] = { 1 };
DBPARAMBINDINFO paramBindInfo[1] = {
     {
           L"DBTYPE_I4",
           NULL,
           sizeof(int),
           DBPARAMFLAGS_ISINPUT,
            0,
          0
     }
};
```

4 Get the ICommandWithParameters interface from the Command object. Set the parameter information for this command:

```
ICommandWithParameters* pi;
hr = pICommandText->QueryInterface(
     IID_ICommandWithParameters, (void**)&pi);
hr = pi->SetParameterInfo(1, rgParamOrdinals,
rgParamBindInfo);
pi->Release();
```

5 The following is a structure that holds the data for all of the parameters. In this case, there is a single int parameter, as shown:

```
struct Parameters {
     int dept_id;
};
```

6 The following array describes the fields in the parameters structure:

```
static DBBINDING ExactBindingsParameters [1] = {
{
     1,           // iOrdinal
     offsetof (Parameters,dept_id), // obValue
     0,           // No length binding
     0,           // No Status binding
     NULL,  // No TypeInfo
     NULL,  // No Object
     NULL,    // No Extensions
     DBPART_VALUE,
     DBMEMOWNER_CLIENTOWNED,   // Ignored
     DBPARAMIO_INPUT,
     sizeof (int),
     0,
     DBTYPE_I4,
     0,              // No Precision
      0                  // No Scale
}
};
```

7 The following interface is the IAccessor interface from the Command object:

```
IAccessor* pIAccessor;
hr = pICommandText->QueryInterface(
     IID_IAccessor, (void**)&pIAccessor);
```

8 Create an accessor on the Command object for the parameters:

```
DBBINDSTATUS status[1];
HACCESSOR hAccessor;
HRESULT hr = pIAccessor-
>CreateAccessor(DBACCESSOR_PARAMETERDATA,
     1, ExactBindingsParameters,
sizeof(ExactBindingsParameters),
     &hAccessor, status);
pIAccessor->Release();
```

9 Create an array of parameters. Each element in the array is a complete set of parameters. The Execute method executes the SQL statement once for each parameter set in the array, as shown:

```
Parameters param = { 1 };
DBPARAMS params[1] = {
     {
          &param,
          1,
           hAccessor
     }

};
```

10 Execute the command:

```
DBROWCOUNT cRowsAffected;
IRowset* pIRowset;
hr = pICommandText->Execute(
     NULL, IID_IRowset, params,
     &cRowsAffected, (IUnknown**)&pIRowset);
```

Executing prepared statements

The ASE OLE DB Provider provides a full set of functions for using prepared statements, which provide performance advantages for statements that are used repeatedly. The following code samples show how to use the prepared statements.

Note To enable compilation and preparation of the statement on ASE, set DynamicPrepare=1.

❖ **To use prepared statements**

1 Get a Command object from the session:

```
ICommandText* pICommandText;
hr = pIDBCreateCommand->CreateCommand(
     NULL, IID_ICommandText,
     (IUnknown**)&pICommandText);
```

2 Set the SQL statement you want to execute:

```
hr = pICommandText->SetCommandText(
     DBGUID_DBSQL,
     L"DELETE FROM department WHERE dept_id = ?");
```

3 Get the ICommandPrepare interface from the Command object. Then, prepare the command by calling Prepare, as shown:

```
ICommandPrepare* pICommandPrepare;
hr = pICommandText->QueryInterface(
     __uuidof(ICommandPrepare),
     (void**)&pICommandPrepare);
hr = pICommandPrepare->Prepare(cExpectedRuns);
pICommandPrepare->Release();
```

4 Create an array to describe the parameters:

```
DB_UPARAMS paramOrdinal[1] = { 1 };
DBPARAMBINDINFO paramBindInfo[1] = {
     {
          L"DBTYPE_I4",
          NULL,
          sizeof(int),
          DBPARAMFLAGS_ISINPUT,
          0,
          0

};
```

5 Get the ICommandWithParameters interface from the Command object and set the parameter information:

```
ICommandWithParameters* pi;
hr = pICommandText->QueryInterface(
     IID_ICommandWithParameters, (void**)&pi);
hr = pi->SetParameterInfo(1, rgParamOrdinals,
rgParamBindInfo);
pi->Release();
```

6 Create a struct to hold the parameter data. This struct contains all of the parameters for this command, as shown:

```
struct Parameters {
     int dept_id;
};
```

The following describes the struct to the command:

```
static DBBINDING ExactBindingsParameters [1] = {
{
     1, // iOrdinal
     offsetof (Parameters,dept_id), // obValue
     0, // No length binding
     0, // No Status binding
     NULL, // No TypeInfo
```

```
          NULL, // No Object
          NULL, // No Extensions
          DBPART_VALUE,
          DBMEMOWNER_CLIENTOWNED, // Ignored
          DBPARAMIO_INPUT,
          sizeof (int),
          0,
          DBTYPE_I4,
          0, // No Precision
          0 // No Scale
     }
};

IAccessor* pIAccessor;
hr = pICommandText->QueryInterface(IID_IAccessor,
(void**)&pIAccessor);

DBBINDSTATUS status[1];
HACCESSOR hAccessor;
HRESULT hr = pIAccessor->CreateAccessor(
     DBACCESSOR_PARAMETERDATA, 1,
     ExactBindingsParameters,
sizeof(ExactBindingsParameters),
     &hAccessor, status);
pIAccessor->Release();

Parameters param = { 1 };
DBPARAMS params[1] = {
     {
          &param,
          1,
          hAccessor
     }
};

DBROWCOUNT cRowsAffected;
IRowset* pIRowset;
hr = pICommandText->Execute(
     NULL, IID_IRowset, params,
     &cRowsAffected, (IUnknown**)&pIRowset);
```

7 Create an accessor for the parameter struct, using the IAccessor interface:

```
IAccessor* pIAccessor;
hr = pICommandText->QueryInterface(IID_IAccessor,
(void**)&pIAccessor);
```

```
DBBINDSTATUS status[1];
HACCESSOR hAccessor;
HRESULT hr = pIAccessor->CreateAccessor(
     DBACCESSOR_PARAMETERDATA, 1,
     ExactBindingsParameters,
sizeof(ExactBindingsParameters),
     &hAccessor, status);
pIAccessor->Release();
```

The following is an array of the parameter sets:

```
Parameters param = { 1 };
DBPARAMS params[1] = {
     {
          &param,
          1,
          hAccessor
     }
};
```

8 Execute the command:

```
DBROWCOUNT cRowsAffected;
IRowset* pIRowset;
hr = pICommandText->Execute(
     NULL, IID_IRowset, params,
     &cRowsAffected, (IUnknown**)&pIRowset);
```

Working with result sets

OLE DB 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, OLE DB applications request this behavior. OLE DB defines a read-only, forward-only cursor, and the ASE OLE DB Provider provides a cursor optimized for performance in this case.

Note To enable server-side cursors, set the UseCursor property to 1.

Retrieving data

The following code example demonstrates how to retrieve data.

❖ **To retrieve data**

1 Create a Command object:

```
ICommandText* pICommandText;
hr = pIDBCreateCommand->CreateCommand(
     NULL, IID_ICommandText,
     (IUnknown**)&pICommandText);
```

2 Set the SQL statement:

```
hr = pICommandText->SetCommandText(
     DBGUID_DBSQL,
     L"SELECT * FROM testReadStringData");
```

3 Create and describe the rowset data structure. This structure contains fields for each column you want accessed, as shown:

```
IAccessor* pIAccessor;
hr = pICommandText->QueryInterface(IID_IAccessor,
(void**)&pIAccessor);

static DBBINDING ExactBindings [1] = {
{
     1, // iOrdinal
     offsetof (ExactlyTheSame,s), // obValue
     0, // No length binding
     0, // No Status binding
     NULL, // No TypeInfo
     NULL, // No Object
     NULL, // No Extensions
     DBPART_VALUE,
     DBMEMOWNER_CLIENTOWNED, // Ignored
     DBPARAMIO_NOTPARAM,
     sizeof(mystr), // number of bytes
     0,
     DBTYPE_WSTR | DBTYPE_BYREF,
     0, // No Precision
     0 // No Scale
}
};

DBBINDSTATUS status[1];
HACCESSOR hAccessor;
HRESULT hr = pIAccessor->CreateAccessor(
          DBACCESSOR_ROWDATA, 1, ExactBindings,
          sizeof(ExactlyTheSame), &hAccessor, status);
     pIAccessor->Release();
```

4 Execute the rowset:

```
DBROWCOUNT cRowsAffected;
IRowset* pIRowset;
hr = pICommandText->Execute(
NULL, IID_IRowset, params,
&cRowsAffected, (IUnknown**)&pIRowset);
```

5 Use the following code to get the rows one row at a time:

```
DBCOUNTITEM cRowsReturned;
HROW hRow[1];
HROW* pRow = hRow;
hr = pIRowset->GetNextRows(NULL, 0, 1,
&cRowsReturned, &pRow);
```

6 Use IMalloc to free the memory allocated by GetData:

```
CComPtr<IMalloc> pIMalloc = NULL;
hr = CoGetMalloc( MEMCTX_TASK, &pIMalloc );

while (hr == S_OK)
{
```

7 Retrieve the data for the specified row, for example:

```
     ExactlyTheSame pData[1] = { {NULL} };
     hr = pIRowset->GetData(hRow[0], hAccessor,
pData);
     wchar_t* value = pData[0].s;
```

8 Free the allocated memory:

```
     // client owned memory must be freed by the
client
     pIMalloc->Free(pData[0].s);
     pData[0].s = NULL;
```

9 Release the rows:

```
     hr = pIRowset->ReleaseRows(1, pRow, NULL, NULL,
NULL);
```

10 Get the next row:

```
     hr = pIRowset->GetNextRows(NULL, 0, 1,
          &cRowsReturned, &pRow);
}
```

```
        pIRowset->Release();
        pICommandText->Release();
```

Get the next row:

```
        hr = pIRowset->GetNextRows(NULL, 0, 1,
              &cRowsReturned, &pRow);
    }

    pIRowset->Release();
    pICommandText->Release();
```

To retrieve rows from a database, execute a SELECT statement using ICommandText::Execute. This opens a cursor on the statement.Then, use IRowset::GetNextRows to fetch rows through the cursor. When an application frees the statement by releasing the rowset, it closes the cursor.

Using scrollable cursors

Scrollable cursors can now move forward and backward within the rowset. Also, Sybase now supports negative values. When a user scrolls backward and forward, the back end provides the corresponding data.

The ASE OLE DB Provider supports the Static Insensitive scrollable cursor. It implements the IRowset::GetNextRows() method, which is a standard method defined in *Microsoft Open Database Connectivity Software Development Kit Programmer's Reference Volume 2*, that is part of the MSDN library. Go to the Microsoft Web site at http://msdn.microsoft.com/library/default.asp?url=/library/en-us/odbc/htm/odbcsqlfetchscroll.asp for more information.

The OLE DB Data Provider supports the following scrolling types:

- Next – return the next row.
- Prior – return the prior row.
- Relative *n* rows – return the row, *n* rows from the current rowset.

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 the ASE version is 15.0 or later. For earlier versions of the ASE server, server-side scrollable cursors is 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.

Setting scrollable cursor attributes

You must set the following attributes to use scrollable cursors:

- DBPROP_CANSCROLLBACKWARDS – if set to VARIANT_TRUE, the rowset allows the lRowsOffset parameter of GetNextRows to be negative.
- DBPROP_CANFETCHBACKWARDS – if set to VARIANT_TRUE, the rowset will allow the cRows parameter of GetNextRows to be negative.

Executing scrollable cursors

❖ **To set up a program to execute a scrollable cursor**

1 Set the scrollable cursor properties on the rowset:

```
DBPROP RowsetProperties[2];
for(int i = 0; i < 2; i++)

        VariantInit(&RowsetProperties[i].vValue);

RowsetProperties[0].dwPropertyID = DBPROP_CANFETCHBACKWARDS;
RowsetProperties[0].vValue.vt     = VT_BOOL;
RowsetProperties[0].vValue.boolVal= VARIANT_TRUE;
RowsetProperties[0].dwOptions = DBPROPOPTIONS_REQUIRED;
RowsetProperties[0].colid         = DB_NULLID;
RowsetProperties[1].dwPropertyID  = DBPROP_CANSCROLLBACKWARDS;
RowsetProperties[1].vValue.vt     = VT_BOOL;
RowsetProperties[1].vValue.boolVal= VARIANT_TRUE;
RowsetProperties[1].dwOptions     = DBPROPOPTIONS_REQUIRED;
RowsetProperties[1].colid         = DB_NULLID;

DBPROPSET rgRowsetPropSet[1];
rgRowsetPropSet[0].guidPropertySet = DBPROPSET_ROWSET;
rgRowsetPropSet[0].cProperties    = 2;
rgRowsetPropSet[0].rgProperties = RowsetProperties;
```

2 Open the rowset:

```
IRowset* pIRowset = ds.OpenRowset("book", 1, rgRowsetPropSet);
```

3 Fetch the rows forward:

```
DBCOUNTITEM cRowsReturned;
HROW hRow[3];
HROW* pRows = hRow;
hr = pIRowset->GetNextRows(NULL, 0, 3, &cRowsReturned, &pRows);
```

4 Release the rows:

```
hr = pIRowset->ReleaseRows(cRowsReturned, pRows, NULL, NULL, NULL);
```

5 Fetch the rows backward:

```
DBCOUNTITEM cRowsReturned;
HROW hRow[3];
HROW* pRows = hRow;
hr = pIRowset->GetNextRows(NULL, 0, -3, &cRowsReturned, &pRows);
```

6 Release the rows:

```
hr = pIRowset->ReleaseRows(cRowsReturned, pRows, NULL, NULL, NULL);
```

7 Release the rowset:

```
pIRowset->Release()
```

Looking at results

To identify the results and the result set interpretation, after you execute a scrollable cursor, refer to the Microsoft MSDN library at http://msdn.microsoft.com/library/default.asp?url=/library/en-us/oledb/htm/oledbirowset__getnextrows.asp

Example of scrollable static insensitive cursor program

For an example of a scrollable, static-insensitive cursor program refer to “Executing scrollable cursors” on page 21.

Calling stored procedures

This section describes how to call stored procedures and process the results from an OLE DB application.

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

❖ **To call stored procedures and process the results**

1 Create a command:

```
ICommandText* pICommandText;
hr = pIDBCreateCommand->CreateCommand(
     NULL, IID_ICommandText,
     (IUnknown**)&pICommandText);
```

2 Set the command's text:

```
hr = pICommandText->SetCommandText(
     DBGUID_DBSQL,
     L"{ call sp_foo(?) }");
```

3 Define the parameters:

```
DB_UPARAMS paramOrdinal[1] = { 1 };
DBPARAMBINDINFO paramBindInfo[1] = {
     {
          L"DBTYPE_I4",
          NULL,
          sizeof(int),
          DBPARAMFLAGS_ISINPUT,
          0,
          0
     }
};
```

4 Set the parameter information on the command:

```
ICommandWithParameters* pi;
hr = pICommandText->QueryInterface(
     IID_ICommandWithParameters, (void**)&pi);
hr = pi->SetParameterInfo(1, rgParamOrdinals,
rgParamBindInfo);
pi->Release();
```

5 Define the parameter's data structure:

```
struct Parameters {
int dept_id;
};

static DBBINDING ExactBindingsParameters [1] = {
     {
          1, // iOrdinal
```

```
                    offsetof (Parameters,dept_id), // obValue
                    0, // No length binding
                    0, // No Status binding
                    NULL, // No TypeInfo
                    NULL, // No Object
                    NULL, // No Extensions
                    DBPART_VALUE,
                    DBMEMOWNER_CLIENTOWNED, // Ignored
                    DBPARAMIO_INPUT,
                    sizeof (int),
                    0,
                    DBTYPE_I4,
                    0, // No Precision
                    0 // No Scale
               }
          };
```

6 Create an accessor for the parameters:

```
IAccessor* pIAccessor;
hr = pICommandText->QueryInterface(IID_IAccessor,
(void**)&pIAccessor);
DBBINDSTATUS status[1];
HACCESSOR hAccessor;
HRESULT hr = pIAccessor->CreateAccessor(
     DBACCESSOR_PARAMETERDATA, 1,
     ExactBindingsParameters,
sizeof(ExactBindingsParameters),
     &hAccessor, status);
pIAccessor->Release();
```

7 Define the parameter data:

```
Parameters param = { 1 };
DBPARAMS params[1] = {
     {
          &param,
          1,
          hAccessor
     }
};

DBROWCOUNT cRowsAffected;
IRowset* pIRowset;
hr = pICommandText->Execute(
     NULL, IID_IRowset, params,
     &cRowsAffected, (IUnknown**)&pIRowset);
```

Handling errors

Errors are reported by returning a failure from a method. All methods return an HRESULT. To determine if a failure has occurred, call FAILED(hr). To get information about the error, call GetErrorInfo.

Example

The following code fragment uses FAILED(hr) and GetErrorInfo:

```
if (FAILED(hr))
{
     IErrorInfo* pIErrorInfo;
     GetErrorInfo(0, &pIErrorInfo);
     BSTR desc;
     pIErrorInfo->GetDescription(&desc);
     // use the desc
     SysFreeString(desc);
     pIErrorInfo->Release();
}
```

Mapping datatypes

The following table describes the ASE OLE DB Provider datatype mappings.

Table 1-2: ASE datatypes and OLE DB datatypes

ASE datatype	OLE DB datatype	C++ datatype
binary	DBTYPE_BYTES	unsigned char[]
bigint	DBTYPE_I8	long long
bit	DBTYPE_BOOL	BOOL
char	DBTYPE_STR, DBTYPE_BSTR	char[], BSTR
date	DBTYPE_DBDATE	DATE_STRUCT
datetime	DBTYPE_DBTIMESTAMP	TIMESTAMP_STRUCT
decimal	DBTYPE_DECIMAL	SQL_NUMERIC
double	DBTYPE_R8	double
float(<16)	DBTYPE_R4	float
float(>=16)	DBTYPE_R8	double

ASE datatype	OLE DB datatype	C++ datatype
image	DBTYPE_IUNKNOWN, DBTYPE_BYTES	IUnknown, unsigned char[] **Note** Sybase recommends that you use streams through IUnknown interfaces. It can also be bound as unsigned char[].
int[eger]	DBTYPE_I4	long
money	DBTYPE_CY	long long
nchar	DBTYPE_STR, DBTYPE_BSTR	char[], BSTR
numeric	DBTYPE_NUMERIC	SQL_NUMERIC
nvarchar	DBTYPE_STR, DBTYPE_BSTR	char[], BSTR
real	DBTYPE_R4	float
smalldatetime	DBTYPE_DBTIMESTAMP	TIMESTAMP_STRUCT
smallint	DBTYPE_I2	short
smallmoney	DBTYPE_CY	long long
text	DBTYPE_IUNKNOWN, DBTYPE_STR	IUnknown, char[] **Note** Sybase recommends that you use streams through IUnknown interfaces. It can also be bound as char[].
time	DBTYPE_DBTIME	TIME_STRUCT
timestamp	DBTYPE_BYTES	unsigned char[]
tinyint	DBTYPE_UI1	unsigned char
unichar	DBTYPE_WSTR, DBTYPE_BSTR	wchar_t[], BSTR
unitext	DBTYPE_IUNKNOWN, DBTYPE_WSTR	IUnknown, wchar_t[] **Note** Sybase recommends that you use IUnknown.
univarchar	DBTYPE_WSTR, DBTYPE_BSTR	wchar_t[], BSTR
unsignedbigint	DBTYPE_UI8	unsigned long long
unsignedint	DBTYPE_UI4	unsigned long
unsignedsmallint	DBTYPE_UI2	unsigned short
varbinary	DBTYPE_BYTES	unsigned char[]
varchar	DBTYPE_STR, DBTYPE_BSTR	char[], BSTR

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 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. For example, you can now have longer names for tables, columns, procedures, and others.

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

CHAPTER 2 Connecting to a Database

This chapter describes how client applications connect to Sybase Adaptive Server Enterprise (ASE) using the ASE OLE DB Provider. It covers the following topics:

Introduction to connections

Any client application that uses ASE 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 OLE DB Provider 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.

How connection parameters work

When an application connects to a database, it uses a set of connection parameters to define the connection, 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:

```
User ID=sa
```

Connection parameters passed as connection strings

Connection parameters are assembled into a connection string, in which a semicolon separates each connection parameter, as shown:

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

The connection string is then passed to the ASE OLE DB Provider.

Using connection parameters

Following is a list of connection parameters that can be supplied to the ASE OLE DB Provider.

Table 2-1: Connection parameters

Property names	Description	Required	Default value
User ID, UserID, UID	A case-sensitive user ID required to connect to the ASE server.	Yes	None
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 the IP address of the ASE server.	No, if data source is specified.	Empty
Port	The port number of the ASE server.	No, if data source is specified.	Empty
AnsiNull	Strict compliance where you cannot use “= NULL.” Instead, you must use “IsNull.”	No	1
ApplicationName	The name ASE uses to identify the client application.	No	Empty
BufferCacheSize	Keeps the input and output buffers in pool. When large results will occur, increase this value to boost performance.	No	20

Property names	Description	Required	Default value
CharSet	The designated character set. The specified character set must be installed on the ASE server. Starting with version 15.0, the charset behavior has changed. If you want to use the ASE server's charset, you must specify it in the connection properties with Charset=ServerDefault. Otherwise, the driver picks up the charset which the client specifies, or gets it from the system environment variables in the order of LC_CTYPE and LANG. If both the environment variables have not been defined, it uses the value iso_1.	No	Empty
ClientHostName	The name of the client host passed in the login record to the server.	No	Empty
ClientHostProc	The identity of the client process on this host machine passed in the login record to the server.	No	Empty
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
DataIntegrity	Enables Kerberos Data Integrity.	No	0 (disabled)
Data Source	The Data Source you want to connect in *Server*:*Port* format.	No, if server and port are specified.	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 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

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 packetsize.	No	1
EncryptedPassword	Specifies if password encryption is enabled: 0 indicates password encryption is disabled, 1 indicates password encryption is enabled.	No	0
Encryption	The designated encryption. Possible values: ssl.	No	Empty
HASession	Specifies if high availability is enabled: 0 indicates high availability disabled, 1 high availability enabled.	No	0
Initial Catalog, Database	The database to which you want to connect.	No	Empty
Language	The language in which ASE returns error messages.	No	Empty – ASE uses English by default.
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)
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: 0 indicates do not enable quoted identifiers, 1 indicates enable quoted identifiers.	No	0
ReplayDetection	Enables Kerberos Replay Detection.	No	0

Property names	Description	Required	Default value
RestrictMaximum PacketSize	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
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. 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.
TrustedFile	If encryption is set to ssl, this property should be set to the path to the Trusted File.	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

Connecting from ADO

Microsoft ActiveX Data Objects (ADO) is an object-oriented programming interface. In ADO, the Connection object represents a unique session with a data source. You can use the following Connection object features to initiate a connection:

- The Provider property holds the name of the provider. If you do not supply a Provider name, ADO uses the MSDASQL provider.
- The *ConnectionString* property holds an ASE connection string. You can supply either OLE DB data source names, or explicit UserID, Password, DatabaseName, and other parameters, just as in other connection strings.

- The Open method uses the connection objects to initiate a connection.

Example

The following Visual Basic code uses the connection objects to initiate an OLE DB connection to ASE:

```
’ Declare the connection object
Dim myConn as New ADODB.Connection
myConn.Provider = "ASEOLEDB"
myConn.ConnectionString ="Data Source=MANGO:5000; User ID=sa"
myConn.Open
```

CHAPTER 3

ASE Advanced Features

This chapter describes the advanced ASE features you can use with the ASE OLE DB Provider. It covers the following topics:

Directory services

Using directory services, the ASE OLE DB Provider can get connection and other information from a central LDAP server 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.

The LDAP server can be located on a different platform than Adaptive Server or the clients. 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 names

See 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.

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, each separated with a semicolon. For example:

```
DSURL={ldap://SYBLDAP:389/dc=sybase,dc=com??one?sybase
Servername=MANGO;
ldap://SYBLDAP1:389/dc=sybase,dc=com??one?sybaseServer
name=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, 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.

Password encryption

By default, the ASE OLE DB Provider sends plain text passwords over the network to ASE for authentication. You can use this feature to change the default 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.

❖ **To encrypt passwords on Windows**

- Set the EncryptPassword property in the connection string to 1.

Data encryption 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.

   Note Transport Layer Security (TSL) is an enhanced version of SSL 3.0, and an alias for the SSL version 3.0 CipherSuites.

2. The server returns its certificate and a list of supported CipherSuites (described in the next section), 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, described next.

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 parameters are specified in the connection string or through directory services such as LDAP.

The ASE OLE DB Provider and Adaptive Server support the CipherSuites that are available with the SSL Plus library API and the cryptographic engine called Security Builder, both from Certicom Corporation.

Note The following list of CipherSuites conform to the TLS specification.

Following is the list of CipherSuites, ordered from strongest to weakest, supported in ASE OLE DB Provider:

- 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 OLE DB Provider

In ASE OLE DB Provider, SSL provides the following levels of security:

- After 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 check 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 OLE DB Provider 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 OLE DB Provider 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," described next.

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 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 OLE DB Provider, add Encryption=*ssl* and TrustedFile=<*filename*> (where *filename* is the path to the *trusted roots* file) to the ConnectString. Then, ASE OLE DB Provider 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*.

❖ **To enable SSL connections on Windows**

1. Set the Encryption property in the connection string to `ssl`.
2. 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.

Kerberos authentication

Kerberos is an industry standard network authentication system that provides simple login authentication as well as mutual login authentication. Kerberos provides user and service authentication. Kerberos 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 and available services.

Adaptive Server and the ASE OLE DB provider provide support for Kerberos connections. The ASE OLE DB provider specifically supports MIT, CyberSafe, and Active Directory KDCs.

Process overview

The Kerberos authentication process works basically 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 install a Kerberos client library. This varies for various operating system vendors.

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

For additional information, refer to vendor documentation.

Enabling Kerberos authentication

To enable Kerberos for the drivers, add the following to your program:

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

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

If you want the Kerberos client to look for the TGT in another cache, you might want to specify the userprincipal method.

If you use SQLDriverConnect with the SQL_DRIVER_NOPROMPT, 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, "AuthenticationClient=mitkerberos;");
strcat(ConnectString, " ServerPrincipal=MANGO;");
```

Windows

Add the following properties to your ConnectionString:

```
AuthenticationClient=<one of activedirectory or
mitkerberos or cybersafekerberos>
ServerPrincipal= <MANGO>
```

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

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 additional 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 To verify that you have a TGT, enter 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