User's Guide

Adaptive Server® Enterprise
OLE DB Provider by Sybase®
12.5.1

[ Microsoft Windows ]
# Contents

About This Book ............................................................................................................................ v

## CHAPTER 1

Introduction to ASE OLE DB Provider ........................................ 1
  Introduction to OLE DB ................................................................. 1
  Supported platforms ....................................................................... 2
  ADO programming with ASE OLE DB Provider ....................... 2
    Connecting to a database using the Connection object .......... 3
    Executing statements using the Command object ................. 4
    Querying the database with the Recordset object............... 4
    Working with the Rowset object .............................................. 6
    Updating data through a cursor .......................................... 6
    Using transactions .................................................................. 7
  Supported OLE DB interfaces...................................................... 8
  ASE OLE DB Provider sample.................................................. 10
  OLE DB programming with ASE OLE DB Provider ............... 11
    Connecting to a data source using OLE DB ....................... 11
    Using threads and connections in OLE DB applications .... 12
  Executing SQL statements......................................................... 12
    Executing statements directly .............................................. 13
    Executing statements with bound parameters ................... 13
    Executing prepared statements ....................................... 16
  Working with result sets ......................................................... 19
    Retrieving data ..................................................................... 19
    Calling stored procedures ................................................ 21
    Handling errors ..................................................................... 23
  Datatype mapping ..................................................................... 24

## CHAPTER 2

Connecting to a Database ......................................................... 27
  Introduction to connections ................................................... 27
  How connection parameters work ........................................ 27
    Passing connection parameters as connection strings ....... 28
    Using connection parameters ......................................... 28
    Connecting from ADO ......................................................... 32
## Contents

**CHAPTER 3**  
ASE Advanced Features ............................................................... 35  
  Directory services ................................................................. 35  
  LDAP as a directory service .................................................... 35  
  Using directory services ....................................................... 36  
  Password encryption ............................................................. 37  
  Data encryption using SSL ..................................................... 38  
  SSL security levels in ASE OLE DB Provider ......................... 40  
  Validating the server by its certificate .................................... 40  
  Enabling SSL connections .................................................... 41

Index ............................................................................................................. 43
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 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 may 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/Maintenance, 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.

v To find the latest information on product certifications

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.

v 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.

2. Click MySybase and create a MySybase profile.

Sybase EBFs and software maintenance

v To find the latest information on EBFs and software maintenance

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:

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

Syntax formatting conventions are summarized in the following table.

<table>
<thead>
<tr>
<th>Table 1: Syntax formatting conventions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Key</strong></td>
</tr>
<tr>
<td>[ ]</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
### 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.

<table>
<thead>
<tr>
<th>Key</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>,</td>
<td>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.</td>
</tr>
<tr>
<td>(</td>
<td>Parentheses are to be typed as part of the command.</td>
</tr>
<tr>
<td>.</td>
<td>An ellipsis (three dots) means you can repeat the last unit as many times as you need. Do not include ellipses in the command.</td>
</tr>
</tbody>
</table>
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:

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction to OLE DB</td>
<td>1</td>
</tr>
<tr>
<td>ADO programming with ASE OLE DB Provider</td>
<td>2</td>
</tr>
<tr>
<td>Supported OLE DB interfaces</td>
<td>8</td>
</tr>
<tr>
<td>ASE OLE DB Provider sample</td>
<td>10</td>
</tr>
<tr>
<td>OLE DB programming with ASE OLE DB Provider</td>
<td>11</td>
</tr>
<tr>
<td>Executing SQL statements</td>
<td>12</td>
</tr>
<tr>
<td>Working with result sets</td>
<td>19</td>
</tr>
<tr>
<td>Calling stored procedures</td>
<td>21</td>
</tr>
<tr>
<td>Handling errors</td>
<td>23</td>
</tr>
<tr>
<td>Datatype mapping</td>
<td>24</td>
</tr>
</tbody>
</table>

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:
ADO programming with ASE OLE DB Provider

- **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 several 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 Windows NT 4.0, 2000, XP, and 2003.

**ADO programming with ASE OLE DB Provider**

ADO (ActiveX Data Objects) 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 the Command1 button to connect and then disconnect.

```vbnet
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:

```cmd
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.

```vba
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 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.

```vba
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.
ADO programming with ASE OLE DB Provider

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:

<table>
<thead>
<tr>
<th>ADO cursor type</th>
<th>ADO constant</th>
<th>ASE type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Static cursor</td>
<td>adOpenStatic</td>
<td>Insensitive cursor</td>
</tr>
<tr>
<td>Forward only</td>
<td>adOpenForwardOnly</td>
<td>No-scroll cursor</td>
</tr>
</tbody>
</table>

Sample code

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

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

Updating data through a cursor

The ASE OLE DB Provider lets you update a result set through a cursor. This capability is not available through the MSDASQL provider.

Updating record sets

You can update the database through a record set.

```vbnet
Private Sub Command6_Click()
Dim myConn As New ADODB.Connection
Dim myRS As New ADODB.Recordset
Dim SQLString As String
' Connect
myConn.Provider = "ASEOLEDB"
myConn.ConnectionString = _
"Data Source=MANGO:5000" _
myConn.Open
myConn.BeginTrans
SQLString = "Select * from customer"
myRS.Open SQLString, _
    myConn, adOpenDynamic, adLockBatchOptimistic
If myRS.EOF And myRS.EOF Then
    MsgBox "Recordset is empty!", _
        16, "Empty Recordset"
```
Else
MsgBox "Cursor type: " + CStr(myRS.CursorType), vbInformation
myRS.MoveFirst
For i = 1 To 3
MsgBox "Row: " + CStr(myRS.Fields("id")), vbInformation
If i = 2 Then
myRS.Update "City", "Toronto"
End If
myRS.MoveNext
Next i
myRS.MovePrevious
myRS.Close
End If
myConn.CommitTrans
myConn.Close End Sub

Notes
If you use the adLockBatchOptimistic setting on the record set, the myRS.Update method does not make any changes to the database itself. Instead, it updates a local copy of the Recordset.

The myRS.UpdateBatch method makes the update to the database server but does not commit it, because it is inside a transaction. If an UpdateBatch method were invoked outside a transaction, the change would be committed.

The myConn.CommitTrans method commits the changes. The Recordset object has been closed by this time, so there is no issue of whether the local copy of the data is changed or not.

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:

<table>
<thead>
<tr>
<th>ADO isolation level</th>
<th>Constant</th>
<th>ASE level</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unspecified</td>
<td>adXactUnspecified</td>
<td>Not applicable. Set to 0</td>
</tr>
<tr>
<td>Chaos</td>
<td>adXactChaos</td>
<td>Unsupported. Set to 0</td>
</tr>
</tbody>
</table>
**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>
<thead>
<tr>
<th><strong>Table 1-1: Supported OLE DB interfaces</strong></th>
<th><strong>Purpose</strong></th>
<th><strong>Limitations</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>IAccessor</td>
<td>Define bindings between client memory and data store values.</td>
<td>DBACESSOR_PASS BYREF not supported. DBACESSOR_OPTIMIZED not supported.</td>
</tr>
<tr>
<td>IColumnsInfo</td>
<td>Get simple information about the columns of a rowset.</td>
<td>NA</td>
</tr>
<tr>
<td>IColumnsRowset</td>
<td>Get information about optional metadata columns in a rowset, and get a rowset of column metadata.</td>
<td>NA</td>
</tr>
<tr>
<td>ICommand</td>
<td>Execute SQL commands.</td>
<td>Does not support calling. ICommandProperties: GetProperties with DBPROPSET_PROPERTIES to find properties that could not have been set.</td>
</tr>
<tr>
<td>ICommandPrepare</td>
<td>Prepare commands.</td>
<td>NA</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>ADO isolation level</strong></th>
<th><strong>Constant</strong></th>
<th><strong>ASE level</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Browse</td>
<td>adXactBrowse</td>
<td>0</td>
</tr>
<tr>
<td>Read uncommitted</td>
<td>adXactReadUncommitted</td>
<td>0</td>
</tr>
<tr>
<td>Cursor stability</td>
<td>adXactCursorStability</td>
<td>1</td>
</tr>
<tr>
<td>Read committed</td>
<td>adXactReadCommitted</td>
<td>1</td>
</tr>
<tr>
<td>Repeatable read</td>
<td>adXactRepeatableRead</td>
<td>2</td>
</tr>
<tr>
<td>Isolated</td>
<td>adXactIsolated</td>
<td>3</td>
</tr>
<tr>
<td>Serializable</td>
<td>adXactSerializable</td>
<td>3</td>
</tr>
</tbody>
</table>
### Interface Purpose Limitations

<table>
<thead>
<tr>
<th>Interface</th>
<th>Purpose</th>
<th>Limitations</th>
</tr>
</thead>
<tbody>
<tr>
<td>ICommandProperties</td>
<td>Set Rowset properties for rowsets created by a command. Most commonly used to specify the interfaces the rowset should support.</td>
<td>NA</td>
</tr>
<tr>
<td>ICommandText</td>
<td>Set the SQL command text for ICommand.</td>
<td>Only the DBGUID_DEFAULT SQL dialect is supported.</td>
</tr>
<tr>
<td>ICommandWithParameters</td>
<td>Set or get parameter information for a command.</td>
<td>No support for parameters stored as vectors of scalar values.</td>
</tr>
<tr>
<td>IConvertType</td>
<td></td>
<td>NA</td>
</tr>
<tr>
<td>IDBCreateCommand</td>
<td>Create commands from a session.</td>
<td>NA</td>
</tr>
<tr>
<td>IDBCreateSession</td>
<td>Create a session from a data source object.</td>
<td>NA</td>
</tr>
<tr>
<td>IDBInfo</td>
<td>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.</td>
<td>NA</td>
</tr>
<tr>
<td>IDBInitialize</td>
<td>Initialize data source objects and enumerators.</td>
<td>NA</td>
</tr>
<tr>
<td>IDBProperties</td>
<td>Manage properties on a data source object or enumerator.</td>
<td>NA</td>
</tr>
<tr>
<td>IDBSchemaRowset</td>
<td>Get information about system tables, in a standard form (a rowset).</td>
<td>NA</td>
</tr>
<tr>
<td>IErrorLookup</td>
<td>Support ActiveX error object.</td>
<td>NA</td>
</tr>
<tr>
<td>IErrorRecords</td>
<td>Support ActiveX error object.</td>
<td>NA</td>
</tr>
<tr>
<td>IGetDataSource</td>
<td>Return an interface pointer to the session's data source object.</td>
<td>NA</td>
</tr>
</tbody>
</table>
### ASE OLE DB Provider sample

A Visual Basic sample that uses the ASE OLE DB Provider is located in the `%SYBASE%\DataAccess\OLEDB\samples` directory.

<table>
<thead>
<tr>
<th>Interface</th>
<th>Purpose</th>
<th>Limitations</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMultipleResults</td>
<td>Retrieve multiple results (rowsets or row counts) from a command.</td>
<td>NA</td>
</tr>
<tr>
<td>IOpenRowset</td>
<td>Access a database table by its name, in a Non-SQL way.</td>
<td>Opening a table by its name is supported, not by a GUID.</td>
</tr>
<tr>
<td>IRowset</td>
<td>Access rowsets.</td>
<td>NA</td>
</tr>
<tr>
<td>IRowsetChange</td>
<td>Allow changes to rowset data, reflected back to the data store.</td>
<td>NA</td>
</tr>
<tr>
<td>IRowsetIdentity</td>
<td>Compare row handles.</td>
<td>NA</td>
</tr>
<tr>
<td>ISequentialStream</td>
<td>Retrieve a blob column.</td>
<td>Supported for reading only. No support for SetData with this interface.</td>
</tr>
<tr>
<td>ISessionProperties</td>
<td>Get session property information.</td>
<td>NA</td>
</tr>
<tr>
<td>ISourcesRowset</td>
<td>Get a rowset of data source objects and enumerators.</td>
<td>NA</td>
</tr>
<tr>
<td>ITableDefinition</td>
<td>Create, drop, and alter tables, with constraints.</td>
<td>NA</td>
</tr>
<tr>
<td>ITransaction</td>
<td>Commit or abort transactions.</td>
<td>Not all the flags are supported.</td>
</tr>
<tr>
<td>ITransactionLocal</td>
<td>Handle transactions on a session.</td>
<td>NA</td>
</tr>
</tbody>
</table>
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 get 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:

```c
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,
                              __uuidof(IDBInitialize), (IUnknown**)&pIDBInitialize);
hr = pIDBInitialize->Initialize();

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

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. Bind parameters are also used with prepared statements to provide performance benefits for statements that are executed more than once.
CHAPTER 1  Introduction to ASE OLE DB Provider

• **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 so improves performance.

### Executing statements directly

The ICommandText::Execute() 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.

Obtain a Command object from the session:

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

Set the SQL statement the command will execute:

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

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:

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

### 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.

Create a Command object from the session:

```c++
ICommandText* pICommandText;
hr = pIDBCreateCommand->CreateCommand(
    NULL, IID_ICommandText,
    (IUnknown**)&pICommandText);
```
Executing SQL statements

Set the SQL statement you want to execute:

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

Create an array to describe the parameters:

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

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

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

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:

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

The following array describes the fields in the parameters structure:

```c
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
};

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

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

Create an accessor on the Command object for the parameters:

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

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
  }
};

Execute the command:

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

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.

**Note** To enable compilation and preparation of the statement on ASE, set the DynamicPrepaare property to 1.

Get a Command object from the session:

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

Set the SQL statement you want to execute:

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

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

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

Create an array to describe the parameters:

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

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();

A struct holds the parameter data. This struct contains all of the parameters for
this command, as shown:

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

The following describes the struct to the command:

```c
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();
Executing SQL statements

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

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

Use the IAccessor interface to create an accessor for the parameter struct:  
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 parameter sets:  
Parameters param = { 1 };  
DBPARAMS params[1] = {  
    &param,  
    1,  
    hAccessor  
};  

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.

Create a Command object:

```csharp
ICMDMText* pCMDMText;
hr = pIDBCMDMCreate->CreateCommand(
    NULL, IID_ICommandText,
    (IUnknown**)&pCMDMText);
```

Set the SQL statement:

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

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

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

static DDBINDING ExactBindings[1] = {
    {1, // iOrdinal
     offsetof(ExactlyTheSame, s), // obValue
     0, // No length binding
     0, // No Status binding
     NULL, // No TypeInfo
```
Working with result sets

```c
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();

Execute the rowset:

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

The following code demonstrates getting the rows one at a time:

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

Use IMalloc to free the memory allocated by GetData:

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

while (hr == S_OK)
{
    Retrieve the data for the specified row:

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

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

Release the rows:

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

Get the next row:

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

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

### 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.

**Example**

Create a command:

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

Set the command’s text:

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

Define the parameters:
Calling stored procedures

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

Set the parameter information on the command:

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

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  
  }  
};
Create an accessor for the parameters:

```c
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();
```

Define the parameter data:

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

```c
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:

```c
if (FAILED(hr))
{
    IErrorInfo* pIErrorInfo;
    GetErrorInfo(0, &pIErrorInfo);
    BSTR desc;
    pIErrorInfo->GetDescription(&desc);
}
Datatype mapping

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

<table>
<thead>
<tr>
<th>ASE datatype</th>
<th>OLE DBTYPE</th>
<th>C++ datatype</th>
</tr>
</thead>
<tbody>
<tr>
<td>binary</td>
<td>DBTYPE_BYTES</td>
<td>unsigned char[]</td>
</tr>
<tr>
<td>bit</td>
<td>DBTYPE_BOOL</td>
<td>BOOL</td>
</tr>
<tr>
<td>char</td>
<td>DBTYPE_STR, DBTYPE_BSTR</td>
<td>char[], BSTR</td>
</tr>
<tr>
<td>date</td>
<td>DBTYPE_DBDATE</td>
<td>DATE_STRUCT</td>
</tr>
<tr>
<td>datatime</td>
<td>DBTYPE_DBTIMESTAMP</td>
<td>TIMESTAMP_STRUCT</td>
</tr>
<tr>
<td>decimal</td>
<td>DBTYPE_DECIMAL</td>
<td>SQL_NUMERIC</td>
</tr>
<tr>
<td>double</td>
<td>DBTYPE_R8</td>
<td>double</td>
</tr>
<tr>
<td>float(&lt;16)</td>
<td>DBTYPE_R4</td>
<td>float</td>
</tr>
<tr>
<td>float(&gt;=16)</td>
<td>DBTYPE_R8</td>
<td>double</td>
</tr>
<tr>
<td>image</td>
<td>DBTYPE_IUNKNOWN, DBTYPE_BYTES</td>
<td>IUnknown, unsigned char[]</td>
</tr>
<tr>
<td>int[eger]</td>
<td>DBTYPE_I4</td>
<td>long</td>
</tr>
<tr>
<td>money</td>
<td>DBTYPE_CY</td>
<td>long long</td>
</tr>
<tr>
<td>nchar</td>
<td>DBTYPE_STR, DBTYPE_BSTR</td>
<td>char[], BSTR</td>
</tr>
<tr>
<td>numeric</td>
<td>DBTYPE_NUMERIC</td>
<td>SQL_NUMERIC</td>
</tr>
<tr>
<td>nvarchar</td>
<td>DBTYPE_STR, DBTYPE_BSTR</td>
<td>char[], BSTR</td>
</tr>
<tr>
<td>real</td>
<td>DBTYPE_R4</td>
<td>float</td>
</tr>
<tr>
<td>smalldatetime</td>
<td>DBTYPE_DBTIMESTAMP</td>
<td>TIMESTAMP_STRUCT</td>
</tr>
<tr>
<td>smallint</td>
<td>DBTYPE_I2</td>
<td>short</td>
</tr>
<tr>
<td>smallmoney</td>
<td>DBTYPE_CY</td>
<td>long long</td>
</tr>
</tbody>
</table>

Note: Sybase recommends you use streams through IUnknown interfaces, but it can also be bound as unsigned char[].
<table>
<thead>
<tr>
<th>ASE datatype</th>
<th>OLE DBTYPE</th>
<th>C++ datatype</th>
</tr>
</thead>
<tbody>
<tr>
<td>text</td>
<td>DBTYPE_IUNKNOWN,</td>
<td>IUnknown, char[]</td>
</tr>
<tr>
<td></td>
<td>DBTYPE_STR</td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Note</strong> Sybase</td>
<td></td>
</tr>
<tr>
<td></td>
<td>recommends you use</td>
<td></td>
</tr>
<tr>
<td></td>
<td>streams through IUnknown</td>
<td></td>
</tr>
<tr>
<td></td>
<td>interfaces, but it can</td>
<td></td>
</tr>
<tr>
<td></td>
<td>also be bound as char[].</td>
<td></td>
</tr>
<tr>
<td>time</td>
<td>DBTYPE_DBTIME</td>
<td>TIME_STRUCT</td>
</tr>
<tr>
<td>timestamp</td>
<td>DBTYPE_BYTES</td>
<td>unsigned char[]</td>
</tr>
<tr>
<td>tinyint</td>
<td>DBTYPE_UI1</td>
<td>unsigned char</td>
</tr>
<tr>
<td>unichar</td>
<td>DBTYPE_WSTR,</td>
<td>wchar_t[], BSTR</td>
</tr>
<tr>
<td></td>
<td>DBTYPE_BSTR</td>
<td></td>
</tr>
<tr>
<td>univarchar</td>
<td>DBTYPE_WSTR,</td>
<td>wchar_t[], BSTR</td>
</tr>
<tr>
<td></td>
<td>DBTYPE_BSTR</td>
<td></td>
</tr>
<tr>
<td>varbinary</td>
<td>DBTYPE_BYTES</td>
<td>unsigned char[]</td>
</tr>
<tr>
<td>varchar</td>
<td>DBTYPE_STR, DBTYPE_BSTR</td>
<td>char[], BSTR</td>
</tr>
</tbody>
</table>
Datatype mapping
CHAPTER 2

Connecting to a Database

This chapter describes how client applications connect to Sybase Adaptive Server Enterprise using the ASE OLE DB Provider.

It covers the following topics:

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction to connections</td>
<td>27</td>
</tr>
<tr>
<td>How connection parameters work</td>
<td>27</td>
</tr>
</tbody>
</table>

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. 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:
User ID=sa

Passing connection parameters 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>
<thead>
<tr>
<th>Property names</th>
<th>Description</th>
<th>Required</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>User ID, UserID, UID</td>
<td>A case-sensitive user ID required to connect to the ASE server.</td>
<td>Yes</td>
<td>None</td>
</tr>
<tr>
<td>PWD, Password</td>
<td>A case-sensitive password to connect to the ASE server.</td>
<td>No, if the user name does not require a password</td>
<td>Empty</td>
</tr>
<tr>
<td>Data Source</td>
<td>The Data Source you want to connect in Server:Port format.</td>
<td>No, if server and port are specified</td>
<td>Empty</td>
</tr>
<tr>
<td>Server</td>
<td>The name or the IP address of the ASE server.</td>
<td>No, if data source is specified</td>
<td>Empty</td>
</tr>
<tr>
<td>Port</td>
<td>The port number of ASE server.</td>
<td>No, if data source is specified</td>
<td>Empty</td>
</tr>
<tr>
<td>Initial Catalog, Database</td>
<td>The database to which you want to connect.</td>
<td>No</td>
<td>Empty</td>
</tr>
<tr>
<td>UseCursor</td>
<td>Specifies whether cursors are to be used by the driver. 0 indicates do not use cursors, and 1 indicates use cursors.</td>
<td>No</td>
<td>0</td>
</tr>
<tr>
<td>ApplicationName</td>
<td>The name ASE uses to identify the client application.</td>
<td>No</td>
<td>Empty</td>
</tr>
<tr>
<td>PacketSize</td>
<td>The number of bytes per network packet transferred between ASE and the client.</td>
<td>No</td>
<td>512</td>
</tr>
<tr>
<td>CharSet</td>
<td>The designated character set. The specified character set must be installed on the ASE server.</td>
<td>No</td>
<td>Empty</td>
</tr>
<tr>
<td>Language</td>
<td>The language in which ASE returns error messages.</td>
<td>No</td>
<td>Empty – ASE uses English by default.</td>
</tr>
</tbody>
</table>
### How connection parameters work

<table>
<thead>
<tr>
<th>Property names</th>
<th>Description</th>
<th>Required</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Encryption</td>
<td>The designated encryption. Possible values: ssl.</td>
<td>No</td>
<td>Empty</td>
</tr>
<tr>
<td>TrustedFile</td>
<td>If encryption is set to ssl, this property should be set to the path to the Trusted File.</td>
<td>No</td>
<td>Empty</td>
</tr>
<tr>
<td>DSURL</td>
<td>The URL to the LDAP server</td>
<td>No</td>
<td>Empty</td>
</tr>
<tr>
<td>DSPPrincipal</td>
<td>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.</td>
<td>No</td>
<td>Empty</td>
</tr>
<tr>
<td>DSPassword</td>
<td>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.</td>
<td>No</td>
<td>Empty</td>
</tr>
<tr>
<td>DynamicPrepare</td>
<td>When set to 1, the driver sends SQLPrepare calls to ASE to compile/prepare. This can boost performance if you reuse the same query repeatedly.</td>
<td>No</td>
<td>0</td>
</tr>
<tr>
<td>Property names</td>
<td>Description</td>
<td>Required</td>
<td>Default value</td>
</tr>
<tr>
<td>-------------------</td>
<td>------------------------------------------------------------------------------------------------</td>
<td>----------</td>
<td>---------------</td>
</tr>
<tr>
<td>LoginTimeOut</td>
<td>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.</td>
<td>No</td>
<td>10</td>
</tr>
<tr>
<td>QuotedIdentifier</td>
<td>Specifies if ASE treats character strings enclosed in double quotes as identifiers. 0 indicates do not enable quoted identifiers, 1 indicates enable quoted identifiers.</td>
<td>No</td>
<td>0</td>
</tr>
<tr>
<td>EncryptedPassword</td>
<td>Specifies if password encryption is enabled. 0 indicates password encryption is disabled, 1 indicates password encryption is enabled.</td>
<td>No</td>
<td>0</td>
</tr>
<tr>
<td>BufferPoolSize</td>
<td>Keeps the input / output buffers in pool. When large results will occur, increase this value to boost performance.</td>
<td>No</td>
<td>20</td>
</tr>
</tbody>
</table>
How connection parameters work

<table>
<thead>
<tr>
<th>Property names</th>
<th>Description</th>
<th>Required</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRC</td>
<td>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.</td>
<td>No</td>
<td>1</td>
</tr>
<tr>
<td>ClientHostName</td>
<td>The name of the client host passed in the login record to the server.</td>
<td>No</td>
<td>Empty</td>
</tr>
<tr>
<td>ClientHostProc</td>
<td>The identity of the client process on this host machine passed in the login record to the server.</td>
<td>No</td>
<td>Empty</td>
</tr>
<tr>
<td>TextSize</td>
<td>The maximum size of binary or text data that will be sent over the wire.</td>
<td>No</td>
<td>Empty. ASE default is 32K.</td>
</tr>
<tr>
<td>AnsiNull</td>
<td>Strict compliance where you cannot use “= NULL.” Instead, you must use “IsNull.”</td>
<td>No</td>
<td>1</td>
</tr>
</tbody>
</table>

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 a connection string. This property holds an ASE connection string. You can supply 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:

```vbnet
' Declare the connection object
Dim myConn as New ADODB.Connection
myConn.Provider = "ASEOLEDB"
myConn.ConnectionString = "Data Source=MANGO:5000; User ID=sa"
myConn.Open
```
How connection parameters work
This chapter describes the advanced ASE features you can use with the ASE OLE DB Provider. It covers the following topics:

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Directory services</td>
<td>35</td>
</tr>
<tr>
<td>Password encryption</td>
<td>37</td>
</tr>
<tr>
<td>Data encryption using SSL</td>
<td>38</td>
</tr>
</tbody>
</table>

**Directory services**

With 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 from the one on which Adaptive Server or the clients are running. 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
Directory services

- High availability companion server name

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. Separate each URL 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 port number, 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, 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 else 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.
  - **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** – 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** TLS, or Transport Layer Security, is an enhanced version of SSL 3.0, and is 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.

### 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, Security Builder, both from Certicom Corp.

**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_DES_CBC_SHA
- TLS_DHE_RSA_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](http://www.ietf.org).

SSL security levels in ASE OLE DB Provider

In ASE OLE DB Provider, 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 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 `ConnectionString`. ASE OLE DB Provider then negotiates an SSL connection with the ASE server.

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

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 filename of the trusted roots file. The filename should contain the path to the file as well.
Data encryption using SSL
Index

A
ADO programming 2
advanced sample 21
ASEOLEDB provider 3

B
bound parameters 13

C
certificate 40
CipherSuites 38
Command object
   executing statements 4
connection
   how parameters work 27
   introduction 27
   setting attributes 12
   strings 28
   table of parameters 29
   using parameters 28
connection functions 11
Connection object
   connecting to a database 3
conventions vii
cursor types 6

data
   retrieving 19
datatype mapping 24
directly executing SQL statements 13
directory services 35
   using 36

DSURL 36

E
EncryptPassword 37
error handling 23
executing prepared statements 16
executing SQL statements 12
executing SQL statements directly 13
executing SQL statements with bound parameters 13

H
handling errors 23
handshake 38

L
LDAP 35

M
MSDASQL 2

O
OLE DB
   interfaces 8
   introduction 1
OLE DB provider 3

User’s Guide
Index

P
| password encryption | 37 |
| prepared statements  | 16 |

Q
| querying             | 4  |

R
| Recordset object     | 4  |
| registering           | 3  |
| result sets           | 19 |
| retrieving data       | 19 |
| return codes          | 23 |
| Rowset object         | 6  |

S
| samples              | 10 |
| advanced             | 21 |
| simple               | 19 |
| Secure Sockets Layer (SSL) | |
| enabling connections | 41 |
| in ASE ODBC Driver   | 40 |
| using                | 38 |
| validation            | 40 |
| setting connections attributes | 12 |
| simple sample         | 19 |
| SQL statements        |    |
| executing             | 12 |
| executing directly    | 13 |
| executing prepared statements | 16 |
| executing with bound parameters | 13 |
| SSL see Secure Sockets Layer | 38 |
| stored procedures     |    |
| calling               | 21 |

T
| threads               | 12 |

transactions    7
trusted roots file 40

V
| validation        | 40 |