SYBASE®

Users Guide

Adaptive Server® Enterprise OLE DB Provider by Sybase

15.5

[Microsoft Windows]

DOCUMENT ID: DC00075-01-1550-01

LAST REVISED: October 2009

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

Contents

About This Book

Audience

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

How to use this book

The information in this book is organized as follows:

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

Related documents

See these books for more information:

- The *Software Developer’s Kit Release Bulletin* for your platform contains important last-minute information about Adaptive Server OLE DB Provider and Software Developer’s Kit (SDK).
- The *Software Developer’s Kit and Open Server Installation Guide* contains information about installing SDK and its Adaptive Server OLE DB Provider component.
- The *Adaptive Server Enterprise Installation Guide* contains information about installing Adaptive Server.
- The *Adaptive Server Enterprise Release Bulletin* for your platform contains information about known problems and recent updates to Adaptive Server.

Other sources of information

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

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

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

  Refer to the *SyBooks Installation Guide* on the Getting Started CD, or the *README.txt* file on the SyBooks CD for instructions on installing and starting SyBooks.
- The Sybase Product Manuals Web site is an online version of the SyBooks CD that you can access using a standard Web browser. In addition to product manuals, you will find links to EBFs/Maintenance, Technical Documents, Case Management, Solved Cases, newsgroups, and the Sybase Developer Network.

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

Sybase certifications on the Web

Technical documentation at the Sybase Web site is updated frequently.

❖ **Finding the latest information on product certifications**

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

❖ **Finding the latest information on component certifications**

1. Point your Web browser to Availability and Certification Reports at http://certification.sybase.com/.
2. Either select the product family and product under Search by Base Product; or select the platform and product under Search by Platform.

3 Select Search to display the availability and certification report for the selection.

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

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

1 Point your Web browser to Technical Documents at http://www.sybase.com/support/techdocs/.

2 Click MySybase and create a MySybase profile.

Sybase EBFs and software maintenance

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

1 Point your Web browser to the Sybase Support Page at http://www.sybase.com/support.

2 Select EBFs/Maintenance. If prompted, enter your MySybase user name and password.

3 Select a product.

4 Specify a time frame and click Go. A list of EBF/Maintenance releases is displayed.

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

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

Conventions

The following conventions are used in this book.

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

  You 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 15.5 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 Adaptive Server OLE DB Provider

This chapter describes how to use the OLE DB interface to get full access to Adaptive Server features from a Microsoft ActiveX Data Objects (ADO) programming environment.

Many applications that use the OLE DB interface do so through the ADO programming model rather than directly. This chapter also describes ADO programming with Adaptive Server.

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 Adaptive Server:

- **Sybase Adaptive Server OLE DB Provider** The Adaptive Server OLE DB Provider has been designed to work with OLE DB 2.5 and later. It provides access to Adaptive Server 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 Adaptive Server ODBC Driver.

Using the Adaptive Server OLE DB Provider brings the following benefits:

- ODBC is not required in your deployment.
- You can get full access to Adaptive Server 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

See the *Open Server and SDK New Features for Windows, Linux, UNIX, and Mac OS X* for a list of platforms on which the Adaptive Server OLE DB Provider is available.

ADO programming with Adaptive Server 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 Adaptive Server OLE DB Provider, you get full access to Adaptive Server 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 Adaptive Server 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:

❖ **Registering the Adaptive Server OLE DB Provider**

1 Open a command prompt.

2 Change to the directory where the Adaptive Server OLE DB Provider is installed.

3 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 Adaptive Server, 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 Adaptive Server is described in the *Adaptive Server Enterprise Transact-SQL Users Guide*.

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

ADO cursor type	ADO constant	ASE type
Static cursor	adOpenStatic	Insensitive cursor
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
```

Updating data through a cursor

The Adaptive Server 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.

```
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;User ID=sa;Pwd=;"
```

```
    myConn.Open
    myConn.BeginTrans
    SQLString = "Select * from customer"
    myRS.Open SQLString, myConn, adOpenDynamic, adLockBatchOptimistic
    If myRS.BOF 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"
                myRS.UpdateBatch
             End If
       myRS.MoveNext
       Next i '
       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 is invoked outside a transaction, the change is 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:

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
Serializable	adXactSerializable	3

Supported OLE DB interfaces

The OLE DB API consists of a set of interfaces. Table 1-1 describes the support for each interface in the Adaptive Server 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.	N/A.
IColumnsRowset	Get information about optional metadata columns in a rowset, and get a rowset of column metadata.	N/A.
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.	N/A.

Interface	Purpose	Limitations
ICommandProperties	Set Rowset properties for rowsets created by a command. Most commonly used to specify the interfaces the rowset should support.	N/A.
ICommandText	Set the SQL command text for ICommand.	Only the DBGUID_DEFAULT SQL dialect is supported.
ICommandWithParameters	Set or get parameter information for a command.	No support for parameters stored as vectors of scalar values.
IConvertType		N/A.
IDBCreateCommand	Create commands from a session.	N/A.
IDBCreateSession	Create a session from a data source object.	N/A.
IDBInfo	Find information about keywords unique to this provider (that is, find nonstandard SQL keywords). Also, find information about literals, special characters used in text matching queries, and other literal information.	N/A.
IDBInitialize	Initialize data source objects and enumerators.	N/A.
IDBProperties	Manage properties on a data source object or enumerator.	N/A.
IDBSchemaRowset	Get information about system tables, in a standard form (a rowset).	N/A.
IErrorLookup IErrorRecords	Support ActiveX error object.	N/A.
IGetDataSource	Return an interface pointer to the session's data source object.	N/A.
IMultipleResults	Retrieve multiple results (rowsets or row counts) from a command.	N/A.
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.	N/A.
IRowsetChange, IRowsetUpdate	Allow changes to rowset data, reflected back to the data store. InsertRow/SetData for blobs not yet implemented.	N/A.
IRowsetIdentity	Compare row handles.	N/A.
ISequentialStream	Retrieve a blob column.	Supported for reading only. No support for SetData with this interface.

Interface	Purpose	Limitations
ISessionProperties	Get session property information.	N/A.
ISourcesRowset	Get a rowset of data source objects and enumerators.	N/A.
ITableDefinition	Create, drop, and alter tables, with constraints.	N/A.
ITransaction	Commit or abort transactions.	Not all the flags are supported.
ITransactionLocal	Handle transactions on a session. Not all the flags are supported.	N/A.

OLE DB programming with Adaptive Server OLE DB Provider

This section describes how to carry out basic tasks in OLE DB while using Adaptive Server 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 Adaptive Server database.

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

- **Connecting to a data source using IDBInitialize**
  1. Call CoCreateInstance.
  2. Pass the clsid obtained from CLSIDFromProgID("ASEOLEDB").
  3. Set the connection properties using IDBInitialize.
- **Connecting to a data source 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 Adaptive Server. 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** Adaptive Server 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.

❖ **Executing 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.

❖ **Constructing and executing 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 Adaptive Server 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 Adaptive Server, set DynamicPrepare=1.

❖ **Using 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 Adaptive Server OLE DB Provider provides a cursor optimized for performance in this case.

To limit the number of rows returned in a row set, use the DBPROP_MAXROWS rowset property. The default value of this property is 0, which indicates no limit to the number of returned rows.

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

Retrieving data

The following code example demonstrates how to retrieve data.

❖ **Retrieving 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();
```

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

IRowset::GetNextRows supports scrollable cursors, which allows both forward and backward movement within the rowset. The rowset moves backward through the results when you specify negative values for the lRowsOffset or cRows parameters of GetNextRows.

The Adaptive Server 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/ for more information.

The OLE DB Provider supports the following scrolling types:

- Next – returns the next row.
- Prior – returns the prior row.
- Relative *n* rows – returns 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 Adaptive Server version is 15.0 or later. For earlier versions of the Adaptive Server, server-side scrollable cursors are not available.
- When the UseCursor connection property is set to 0, client-side scrollable cursors (cached result sets) are used, regardless of the Adaptive Server 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

❖ **Executing 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/

Example of scrollable static insensitive cursor program

For an example of a scrollable, static-insensitive cursor program refer to “Executing scrollable cursors” on page 22.

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 *Adaptive Server Enterprise Reference Manual*.

❖ **Calling stored procedures and processing 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 Adaptive Server OLE DB Provider datatype mappings.

Table 1-2: Adaptive Server datatypes and OLE DB datatypes

ASE datatype	OLE DB datatype	C++ datatype
bigdatetime	DBTYPE_DBTIMESTAMP	TIMESTAMP_STRUCT
bigint	DBTYPE_I8	long long
bigtime	DBTYPE_DBTIME	TIME_STRUCT
binary	DBTYPE_BYTES	unsigned char[]
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
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[].

ASE datatype	OLE DB datatype	C++ datatype
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 Adaptive Server OLE DB Provider supports 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 Adaptive Server OLE DB Provider supports object names or identifiers that have lengths of up to 255 bytes. For information on Adaptive Server large identifiers, see *What’s New in Adaptive Server Enterprise 15.0?*

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

Adaptive Server OLE DB Provider sample

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

OLE DB DSN Migration

You can use the OLE DB DSN migration tool to migrate your data source definitions (DSNs) from the OLE DB Driver Kit to the Adaptive Server OLE DB Provider by Sybase. The migrated DSNs will use the Adaptive Server OLE DB Provider by Sybase instead of the OLE DB Driver Kit.

Migrating to Adaptive Server OLE DB Provider by Sybase

To migrate OLE DB applications to use the Adaptive Server OLE DB Provider by Sybase, edit the connection string used by OLE DB client applications. The provider short name for the Adaptive Server OLE DB Provider is "ASEOLEDB."

Note The Adaptive Server OLE DB Provider connection string syntax differs from that of OLE DB Driver Kit. Although the OLE DB Driver Kit syntax is supported, Sybase recommends that you migrate your connection string syntax to the Adaptive Server OLE DB Provider syntax when possible.

Migrating Data Source Names to Sybase drivers

There are two methods to migrate data source names (DSNs) from the OLE DB Driver Kit to the drivers created by Sybase:

- Using the Sybase Adaptive Server Data Source Administrator
- Using the DSN migration tool

Using the Sybase Adaptive Server Data Source Administrator

The Sybase Adaptive Server Data Source Administrator allows you to migrate existing OLE DB Driver Kit data sources, and to create new data sources for the Adaptive Server OLE DB Provider.

❖ **Migrating data sources using the Data Source Administrator**

1 On the main window titled "Sybase Data Source Administrator," choose the data source.

2 Click Migrate.

The Sybase Data Source Administrator allows you to add, remove, configure, or test the OLE DB data sources.

Using the DSN migration tool

The DSN migration tool can help you migrate the data sources from the OLE DB Driver Kit to the OLE DB Driver by Sybase.

The dsnmigrate tool uses switches to control which DSNs are migrated. From the command line, enter:

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

For all converted OLE DB DSNs, the new Sybase DSNs will have the same name.

Conversion switches

Table 1-3 lists and describes the switches used in the conversion.

Table 1-3: Conversion switches

Switches	Description of results
/?,/h,/help	Displays available dsnmigrate switches. These switches also appears if dsnmigrate is called with no command line arguments.
/oledb	Places dsnmigrate into OLEDB-mode. By default, ODBC DSNs are migrated.
/l	Displays a list of all OLE DB Driver Kit user and system DSNs.
/ul	Displays a list of all OLE DB Driver Kit user DSNs.
/sl	Displays a list of all OLE DB Driver Kit system DSNs.
/a	Converts all OLE DB Driver Kit user and system DSNs.
/ua	Converts all OLE DB Driver Kit user DSNs.
/sa	Converts all OLE DB Driver Kit system DSNs.

Switches	Description of results
/dsn	Converts specific OLE DB Driver Kit user or system DSNs.
/udsn	Converts specific OLE DB Driver Kit user DSNs.
/sdsn	Converts specific OLE DB Driver Kit system DSNs.
dsn	The name of the DSN to be converted.
/suffix	An optional switch that changes the way DSNs are named. If this switch is used, the original DSN is retained and the new DSN is named "<dsn>-<suffix>."
suffix	The suffix that is used to name the new DSN.

CHAPTER 2

Connecting to a Database

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

Introduction to connections

Any client application that uses Adaptive Server 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 Adaptive Server 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 Adaptive Server server running the required database.

Installing OLE DB MetaData stored procedures

You must install the OLE DB MetaData stored procedures on any Adaptive Server that you want to connect to using the OLE DB Provider.

❖ **Installing OLE DB stored procedures on Adaptive Server**

1 Change to the *sp* directory under the OLE DB installation directory.

- For OLE DB 32-bit: *%SYBASE%\DataAccess\oledb\sp*
- For OLE DB 64-bit: *%SYBASE%\DataAccess64\oledb\sp*

2 Execute the *install_oledb_sprocs* script:

```
install_oledb_sprocs ServerName username [password]
```

where:

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

How connection parameters work

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

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

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

Saving connection parameters in OLE DB data sources

When connecting to the database, an OLE DB application can use OLE DB data sources, which are sets of connection parameters stored in a file.

Use the Sybase Data Source Administrator to configure OLE DB data sources.

Connecting using a data source

OLE DB applications typically use data sources on the client computer for each database you want to connect to. You can store sets of Adaptive Server Enterprise connection parameters as an OLE DB data source. If you have a data source, your connection string can simply name the data source by using the DataSource connection parameter:

```
Data Source=my data source;
```

Using connection parameters

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

Table 2-1: Connection parameters

Property names	Description	Required	Default value
AnsiNull	Strict compliance where you cannot use "= NULL." Instead, you must use "IsNull."	No	1
ApplicationName	The name Adaptive Server 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
CharSet	Specifies the character set that is used to communicate to Adaptive Server. By default, the Adaptive Server OLE DB Driver negotiates the same default character set as the Adaptive Server.	No	Empty
ClientCharset	Specifies the client character set. In cases where ClientCharset is different from CharSet, the Adaptive Server OLE DB Provider converts the client's character set to the character set specified by CharSet.	No	The character set currently used by the operating system.
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
CodePageType	Specifies the type of character encoding used. The valid values are ANSI and OEM.	No	ANSI

Property names	Description	Required	Default value
CRC	By default, the driver returns the total records updated when multiple update statements are executed in a stored procedure. This count will also include all updates happening as part of the triggers set on an update or an insert. Set this property to 0 if you want the driver to return only the last update count.	No	1
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
DynamicPrepare	When set to 1, the driver sends SQLPrepare calls to Adaptive Server to compile/prepare. This can boost performance if you use the same query repeatedly.	No	0
EnableServerPacketSize	Allows Adaptive Server 15.0 or later to choose the optimal packetsize.	No	1
EncryptedPassword	Specifies whether password is transmitted in an encrypted format: • 0 – use plain text password. • 1 – use encrypted password. If it is not supported, return an error message. • 2 – use encrypted password. If it is not supported, use plain text password. **Note** When password encryption is enabled, and the server supports asymmetric encryption, asymmetric encryption is used instead of symmetric encryption.	No	0

Property names	Description	Required	Default value
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 Adaptive Server returns error messages.	No	Empty – Adaptive Server 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	15
MutualAuthentication	Enables Kerberos Mutual Authentication.	No	0 (disabled)
oldpassword	The current password. If oldpassword contains a value that is not null or an empty string, the current password is changed to the value contained in PWD.	No	Empty
PacketSize	The number of bytes per network packet transferred between Adaptive Server and the client.	No	Server-determined when driver is connected to Adaptive Server 15.0 or later. For older Adaptive Server the default is 512.
Port	The port number of the Adaptive Server server.	No, if data source is specified.	Empty
PWD, Password	Contains the value of the password. When performing a normal login, oldpassword is not set and PWD contains the value of the current password. When changing the password, oldpassword is set to the current password and PWD contains the value of the new password.	No, if the user name does not require a password.	Empty

Property names	Description	Required	Default value
QuotedIdentifier	Specifies if Adaptive Server 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
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 Adaptive 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 Adaptive Server acting as a failover server in an active-active or active-passive setup.	Yes, if HASession is set to 1	Empty
Server	The name or the IP address of the Adaptive Server.	No, if data source is specified.	Empty
ServerInitiated Transactions	When ServerInitiatedTransactions is set to 1, Adaptive Server starts managing transactions as needed. The driver issues a set chained on command on the connection. Set this property to 0 if you 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. Adaptive Server 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
User ID, UserID, UID	A case-sensitive user ID required to connect to the Adaptive Server server.	Yes	None

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 Adaptive Server 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 Adaptive Server:

```
' 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

Supported Adaptive Server Features

This chapter describes the Adaptive Server features you can use with the Adaptive Server OLE DB Provider.

Microsecond granularity for time data

The Adaptive Server OLE DB Provider provides microsecond-level precision for time data by supporting the SQL datatypes bigdatetime and bigtime.

bigdatetime and bigtime function similarly to and have the same data mappings as the SQL datetime and time datatypes:

- bigdatetime corresponds to the Adaptive Server bigdatetime datatype and indicates the number of microseconds that have passed since January 1, 0000 0:00:00.000000. The range of legal bigdatetime values is from January 1, 0001 00:00:00.000000 to December 31, 9999 23:59:59.999999.

- bigtime corresponds to the Adaptive Server bigtime datatype and indicates the number of microseconds that have passed since the beginning of the day. The range of legal bigtime values is from 00:00:00.000000 to 23:59:59.999999.

Usage

- When connecting to Adaptive Server 15.5, the Adaptive Server OLE DB Provider transfers data using the bigdatetime and bigtime datatypes even if the receiving Adaptive Server columns are defined as datetime and time.

  This means that Adaptive server may silently truncate the values from the Adaptive Server OLE DB Provider to fit Adaptive Server columns. For example, a bigtime value of 23:59:59.999999 is saved as 23:59:59.996 in an Adaptive Server column with datatype time.
- When connecting to Adaptive Server 15.0.x and earlier, the Adaptive Server OLE DB Provider transfers data using the datetime and time datatypes.
- Earlier versions of Adaptive Server OLE DB Provider continue to use the datetime and time datatypes.

Supported Adaptive Server Cluster Edition features

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

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

Login redirection

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

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

Connection migration

Connection migration allows a server in a Cluster Edition environment to dynamically distribute load, and seamlessly migrate an existing client connection and its context to another server within the cluster. This feature enables the Cluster Edition environment to achieve optimal resource utilization and decrease computing time. Because migration between servers is seamless, the connection migration feature also helps create a truly high availability (HA), zero-downtime environment. Connection migration is enabled automatically when a client application connects to a server that supports this feature.

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

Connection failover enhancement

Existing connection failover allows a client application to switch to an alternate Adaptive Server if the primary server becomes unavailable due to an unplanned event, like power outage or a socket failure. This feature has been enhanced to allow client applications to fail over numerous times to multiple servers using dynamic failover addresses.

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

Enabling Cluster Edition connection failover

Using the OLEDB user interface

One way to enable the extended connection failover in Adaptive Server OLE DB Provider is through its user interface.

❖ **Using the user interface to enable extended failover**

1 Open the Configure Data Source dialog box of the Sybase Datasource Administrator.

2 Go to the Connection tab.

3 Select Enable High Availability.

4 Optionally, enter the alternate servers and ports in the Alternate Servers field using the format:

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

OLE DBIn establishing a connection, the Adaptive Server Provider first attempts to connect to the primary host and port defined in the General tab of the Configure Data Source dialog box. If the Adaptive Server OLE DB Provider fails to establish a connection, OLE DB goes through the list of hosts and ports specified in the Alternate Servers field.

Using the OLEDB connection string

You can also use the OLE DB connection string to enable extended failover by setting the HASession connection string property to 1. For example:

```
Provider=ASEOLEDB;User ID=sa;Password=;
InitialCatalog=sdc;Data Source=server1:port1;
ProviderString='HASession=1;AlternateServers=
server2:port2,...,serverN:portN';
```

In the preceding example, Data Source defines the primary server and port. The Adaptive Server OLE DB Provider attempts to establish connection to the primary server first. If unsuccessful, and alternate servers are defined, OLE DB goes through the servers listed in the Alternate Servers field until a connection is established or until the end of the list is reached.

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

Directory services

Using directory services, the Adaptive Server OLE DB Provider can get connection and other information from a central LDAP server to connect to an Adaptive 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 the *Adaptive Server Enterprise System Administration Guide* for more information.

The LDAP server can be configured with these access restrictions:

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

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

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.

Interfaces file

When you use an interfaces file it appears similar to any of the following:

- `file///mils1/sybase/ini/sql.ini`
- `file:///c:/sybase/ini/sql.ini`
- `file://c:\sybase\ini\sql.ini`
- `file:///sql.ini`

In this case, the URL is a file and resolves to the location of the interfaces file. By default the Adaptive Server OLE DB Provider assumes a complete file URL for the interfaces file. If a complete path is not specified, the driver looks for that file in the *$SYBASE* tree structure, the default location of the interfaces file for all Sybase products.

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 Adaptive Server OLE DB Provider sends plain text passwords over the network to Adaptive Server for authentication. However, Adaptive Server OLE DB Provider also supports symmetrical and asymmetrical password encryption; you can change the default behavior and encrypt your passwords before they are sent over the network.

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

You can encrypt login and remote passwords using the Sybase Common Security Infrastructure (CSI). CSI 2.6 complies with the Federal Information Processing Standard (FIPS) 140-2.

Enabling password encryption

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

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

- 2 – use encrypted password. If it is not supported, use plain text password.

Note To use asymmetrical password encryption, your server must support asymmetrical encryption, such as Adaptive Server 15.0.2. Asymmetrical encryption requires additional processing time, and may cause a slight delay in login time.

❖ **Encrypting passwords**

1 Launch the Sybase Data Source Administrator.

2 Select Data Source.

3 Check Encrypt Password.

Note You can only use the user interface to set EncryptPassword to 0 or 1. To set EncryptPassword to 2, use a connection string.

Example

```
Data Source=MANGO:5000;UserID=sa;pwd=sybase;EncryptPassword=2
```

Password expiration handling

Every company has a specific set of password policies for its database system. Depending on the policies, the password expires at a specific date and time. Unless the password is reset, the Adaptive Server drivers connected to a database throw password expired errors and suggest that the user change the password using isql. The password change feature enables users to change their expired passwords without having to use another tool.

To change your password, set these two connection properties:

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

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 Adaptive Server OLE DB Provider and Adaptive Server support the cipher suites 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 Adaptive Server 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 Adaptive Server OLE DB Provider

In Adaptive Server 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 Adaptive Server 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). Adaptive Server 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 Adaptive Server OLE DB Provider, add Encryption=*ssl* and TrustedFile=<*filename*> (where *filename* is the path to the *trusted roots* file) to the ConnectString. Then, Adaptive Server OLE DB Provider negotiates an SSL connection with Adaptive Server.

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

❖ **Enabling SSL connections on Microsoft 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.

Bookmark and batch operation support for OLE DB

Sybase supports bookmarks and SQL batch operations for the OLE DB Provider.

Bookmark operations use the IRowsetLocate interface to provide methods for comparing bookmarks and retrieving rows based on bookmarks. Batch operations are supported through the IRowsetUpdate interface, which provides methods for updating, inserting, and deleting batches of rows. For instructions about using IRowsetLocate and IRowsetUpdate, see the Microsoft Developer Network at http://msdn.microsoft.com.

HA failover on Adaptive Server OLE DB Provider

Table 3-1 lists the Adaptive Server OLE DB Provider connection parameters used for high availability (HA) failover:

Table 3-1: HA failover connection parameters

Property names	Description	Required	Default value
HASession	Specifies if high availability is enabled. 0 indicates high availability disabled, 1 high availability enabled.	No	0
SecondaryPort	The port number of the Adaptive 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 Adaptive Server acting as a failover server in an active-active or active-passive setup.	Yes, if HASession is set to 1.	Empty

Using failover in HA systems

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

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

For example, Adaptive Server 1, the primary companion server, fails. Adaptive Server 2, as the secondary companion server, reads its disks (disks 1 – 4) and manages any databases on them until Adaptive Server 1 can be restarted. Any clients connected to Adaptive Server 1 are automatically connected to Adaptive Server 2.

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

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

When the Adaptive Server OLE DB Provider detects a connection failure with the primary Adaptive Server, it first tries to reconnect to the primary. If it cannot reconnect, it assumes that a failover has occurred. Then, it automatically tries to connect to the secondary Adaptive Server using the connection properties set in SecondaryServer, and SecondaryPort.

Confirming a successful failover

If a connection to the secondary Adaptive Server is established, the Adaptive Server OLE DB Provider returns "E_FAIL" for the function return HRESULT.

To confirm a successful failover, examine the dwMinor field in ERRORINFO (returned from IErrorRecords::GetBasicErrorInfo), or the description returned from IErrorInfo::GetDescription. The dwMinor value should be "30130" for a successful HA failover. The description from IErrorInfo::GetDescription should be as follows, where *ASEServerName* is the server name failed over to:

```
"Sybase server is not available or has terminated your
connection, you have successfully connected to the next
available HA server ASEServerName. All transactions has
been rolled back."
```

Note Sybase recommends that you check the code returned by dwMinor to determine the success of the failover rather than through examination of the error description.

The client must then reapply the failed transaction with the new connection. If failover happens while a transaction is open, only the changes that were committed to the database before failover are retained.

Verifying an unsuccessful failover

If the connection to the secondary server is not established, the Adaptive Server OLE DB Provider also returns "E_FAIL" for the function return HRESULT. However, the dwMinor field in ERRORINFO (returned from IErrorRecords::GetBasicErrorInfo) should be "30131", and the description returned from IErrorInfo::GetDescription should be:

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

Sample code for checking failover

The following code snippet shows how to code for a failover:

```
/* Declare required variables */
...
/* Open Database connection */
...
/* Perform a transaction */
...
/*Check HRESULT  and dwMinor in ERRORINFO, handle failover */
if (FAILED(hr))
{
   IErrorInfo* pIErrorInfo;
   GetErrorInfo(0, &pIErrorInfo);
   IErrorRecords * pIErrorRecords;
   HRESULT hr1 = pIErrorInfo->QueryInterface(IID_IErrorRecords,
      (void **)&pIErrorRecords);
   if (SUCCEEDED(hr1))
   {
      ERRORINFO errorInfo;
      pIErrorRecords->GetBasicErrorInfo(0, &errorInfo);
      pIErrorRecords->Release();
      if (errorInfo.dwMinor == 30130)
      {
         //successful failover,
         //retry the transaction
      }
   }
}
```

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.

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

Adaptive Server and the Adaptive Server OLE DB Provider support Kerberos connections. The Adaptive Server 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 Microsoft Windows, the Windows Active Directory client library comes installed with the client library.
- CyberSafe and MIT client libraries are available for Microsoft Windows.

For additional information, see the endor 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 use this information to negotiate Kerberos authentication with the configured KDC and Adaptive 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;");
```

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.

❖ **Generating 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. This ticket is valid for approximately 8 hours.

Adaptive Server OLE DB Provider participation in distributed transactions

This feature requires that Microsoft Distributed Transaction Coordinator (MS DTC) be the transaction coordinator managing distributed transactions.

Sybase supports these programming models:

- Applications using MS DTC directly.
- Applications using Microsoft Transaction Server (MTS) or (COM+).

Programming for MS DTC

❖ **Programming using Microsoft Distributed Transaction Coordinator (MS DTC)**

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

2 Get the IDBSession for each Adaptive Server connection you want to establish following the OLE DB steps.

3 Call the ITransactionDispenser::BeginTransaction function to begin an MS DTC transaction and to obtain an OLE Transaction object that represents the transaction.

4 Query ITransactionJoin from each IDBSession (OLE DB Connection) you want to enlist in the MS DTC transaction, and call JoinTransaction with the passed in parameter punkTransactionCoord as the Transaction object (obtained in Step 3). Currently, Sybase supports only the isolation level of ISOLATION LEVEL_READCOMMITTED for the distributed transaction, and does not support ITransactionOptions.

5 To update SQL Server, follow the OLE DB steps for creating and executing IDBCommand.

6 Call the ITransaction::Commit function to commit the MS DTC transaction. The Transaction object is no longer valid.

Programming components deployed in MTS or COM+

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

❖ **Programming components deployed in MTS or COM+**

1 Create an IDBSession for each Adaptive Server connection.

2 Create and execute IDBCommand for each update you would like to perform.

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

The COM+, OLE DB Services, and OLE DB provider will take care of creating the transaction, participating in the transaction, and committing or rolling back the transaction.

OLE DB Services is needed for the Automatic Transaction Enlistment. To enable the OLE DB Services, you must follow some rules to initialize the Data Source (see the MS OLE DB documents). To enable the automatic transaction enlistment, you can set the bit DBPROPVAL_OS_TXNENLISTMENT in the *OLE_DB_SERVICES* registry and in the DBPROP_INIT_OLEDBSERVICES property value, or pass OLE DB Services = 2 in the connection string.

Connection properties for Distributed Transaction support

The following describes the connection properties:

- Distributed Transaction Protocol (DistributedTransactionProtocol) – to specify the protocol used to support the distributed transaction, either use the XA Interface standard or MS DTC OLE Native protocol, select the Distributed Transaction Protocol in the OLE DB Data Source Dialog, set the property DistributedTransactionProtocol = *OLE* in the provider string part of the connection string for OLE Native protocol, or the use default protocol *XA*.
- Tightly Coupled Transaction (TightlyCoupledTransaction) – when a distributed transaction using two resource managers points to the same Adaptive Server, you may have a situation called “Tightly Coupled Transaction.” Under these conditions, if you do not set this property to 1, the Distributed Transaction may fail.

To summarize, if you open two database connections to the same Adaptive Server and then enlist these connections in the same distributed transaction, Sybase recommends that you set TightlyCoupledTransaction=1. To set this property, select the Tightly Coupled Transaction in the OLE DB Data Source dialog box, or pass the property TightlyCoupledTransaction=1 in the provider string part of the connection string.

Index

M

N

O

P

Q

R

S

T

V