SYBASE®

Users Guide

Adaptive Server® Enterprise ADO.NET Data Provider

1.155

[Microsoft Windows]

DOCUMENT ID: DC20066-01-0115-02

LAST REVISED: October 2009

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

Contents

About This Book

Audience

This document is intended for application developers who need access to data from Adaptive Server® Enterprise using any of the supported .NET programming languages. To use this book, you must be familiar with the Microsoft ADO.NET technology and able to code to the ADO.NET specification.

How to use this book

The information in this book is organized as follows:

- Chapter 1, “Understanding and Deploying Adaptive Server Enterprise ADO.NET Data Provider,” introduces you to the Adaptive Server ADO.NET Data Provider.
- Chapter 2, “Using the Sample Applications,” contains information about using the sample projects included with Adaptive Server ADO.NET Data Provider.
- Chapter 3, “Developing Applications,” contains information about developing and deploying applications with the Adaptive Server ADO.NET Data Provider.
- Chapter 4, “Adaptive Server Advanced Features,” contains information about the Adaptive Server features that you can use with Adaptive Server ADO.NET Data Provider.
- Chapter 5, “Adaptive Server ADO.NET Data Provider API Reference,” contains information about the Adaptive Server ADO.NET Data Provider APIs.

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 ADO.NET Data 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 ADO.NET Data 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.

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

  You can use an Insert, Update, or Delete statement with the ExecuteNonQuery method.

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

  The set password *new_passwd* clause specifies a new password.

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

  The value of the pubs2 object.

- Syntax statements that display the syntax and options for a command are printed as follows:

  AseDataAdapter *adapter*
  string *connectionString*
  AseCommand *selectCommand*

  Examples that show the use of commands are printed as follows:

```
AseConnection conn = new AseConnection(
   "Data Source=’mango’;" +
   "Port=5000;" +
   "UID=’sa’;" +
   "PWD='';" +
   Database='pubs2';" );
```

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 may choose or omit enclosed options. Do not include brackets in the command.
|	Vertical bars mean you may choose no more than one option (enclosed in braces or brackets).
,	Commas mean you may 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 may also be required in other syntax contexts.
( )	Parentheses are to be typed as part of the command.
. . .	An ellipsis (three dots) means you may 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.

Adaptive Server Enterprise ADO.NET Data Provider documentation has 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.

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 MixedCase 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

Understanding and Deploying Adaptive Server Enterprise ADO.NET Data Provider

This chapter introduces you to the Adaptive Server Enterprise ADO.NET Data Provider.

What is Adaptive Server ADO.NET Data Provider?

Adaptive Server ADO.NET Data Provider is an ADO.NET provider for the Sybase Adaptive Server database. It is supported on Adaptive Server versions 12.5.x, 15.0.x, and 15.5.x, and all Adaptive Server CE versions.

Adaptive Server ADO.NET Data Provider allows you to access data in Adaptive Server using any language supported by .NET, such as C#, Visual Basic .NET, C++ with managed extensions, and J#. It is also a .NET common language runtime (CLR) assembly, which is a class library that contains all the required sets of classes that provide functionality for all the ADO.NET interfaces. All the classes are managed code and accessible from any managed client code. This inter-language communication is provided by the Microsoft .NET framework.

Some key benefits to using Adaptive Server ADO.NET Data Provider are:

- Adaptive Server ADO.NET Data Provider is faster than the OLE DB Provider.

- In the .NET environment, Adaptive Server ADO.NET Data Provider provides native access to Adaptive Server. Unlike other supported providers, it communicates directly with Adaptive Server and does not require bridge technology.

Deploying Adaptive Server ADO.NET Data Provider

The following sections describe the requirements for deploying Adaptive Server ADO.NET Data Provider. For a list of supported platforms, see the Sybase platform certifications page at http://certification.sybase.com/ucr/search.do.

System requirements

To use Adaptive Server ADO.NET Data Provider, you must have the following installed on your machine:

- **For development –** NET Framework SDK 1.1 and Visual Studio .NET 2003 or a .NET language compiler, such as C#
- **For deployment –** NET Framework 1.1

Required files

Adaptive Server ADO.NET Data Provider consists of these files:

- *Sybase.Data.AseClient.dll* is the provider assembly referenced by the client code.
- *sbgse2.dll*, *sybcsi_certicom_fips26.dll*, *sybcsi_core26.dll*, and *sybcsi_profiler26.dll* contain code for SSL support.
- *sybdrvado115.dll* contains utility code.
- *sybdrvkrb.dll* contains code for Kerberos authentication.

Deploying Adaptive Server ADO.NET Data Provider assembly into the global assembly cache

Multiple applications on a single machine often share the Adaptive Server ADO.NET Data Provider assembly. This can result in duplicate copies of the assembly, and compatibility and version control problems. To avoid this, Sybase recommends that you deploy the Adaptive Server ADO.NET Data Provider assembly into the global assembly cache (GAC), a machine-wide cache that stores and manages assemblies shared by several applications on the same machine. If this is not possible, install copies of the Adaptive Server ADO.NET Data Provider assembly in all the directories where applications using the provider will execute.

The Adaptive Server ADO.NET Data Provider installation program automatically deploys the assembly into the GAC if it detects that the .NET Framework SDK 1.1 is installed on the system. If you do not have the SDK installed or did not use the installation program, the assembly needs to be manually deployed. You can do this by using the .NET Framework Configuration tool.

❖ **Deploying the assembly using the .NET Framework Configuration tool**

1. Start the .NET Framework Configuration tool. Refer to the Microsoft documentation for your specific operating system for instructions on how to start the configuration tool.
2. Select Assembly Cache from the tree view on the left.
3. Click Add an Assembly to the Assembly Cache link on the panel.
4. In the Add an Assembly dialog box, find Adaptive Server ADO.NET Data Provider assembly located in the installation directory and click open. The default installation directory is *C:\Sybase\DataAccess\ADONET\dll*.

   Adaptive Server ADO.NET Data Provider assembly is now deployed into the GAC. To verify this, select the View List of Assemblies in the Assembly Cache link from the panel and examine the list of assemblies in the cache.

Removing the assembly from GAC

You can remove the assembly from the GAC by using the .NET Framework Configuration tool.

❖ **Removing the assembly using the .NET Framework Configuration tool**

1. Start the .NET Framework Configuration tool. Refer to the Microsoft documentation for your specific operating system for instructions on how to start the configuration tool.
2. Select Assembly Cache from the tree view on the left.
3. Click the View List of Assemblies in the Assembly Cache link on the panel.
4. Find *Sybase.Data.AseClient* from the list of Assembly names. There may be multiple entries of this assembly corresponding to various versions deployed on this system.
5. Select one or more assemblies you want to remove. Right-click and select Delete. Click Yes to confirm.
6. Check for the Publisher policy file corresponding to the versions removed and remove these files too.

Note The GAC stores references made by other assemblies to a given assembly and you cannot delete the given assembly until these references are removed. You can force these references to be removed as part of the delete command. On some systems, the utility might fail to delete the assembly and complain about a pending reference to Windows Installer. This happens due to some residual values in the registry. Contact Microsoft support to get a resolution to this problem.

Deploying applications that use Adaptive Server ADO.NET Data Provider

The procedures below demonstrate how to deploy applications.

❖ **Using the installation program and GAC to deploy an application**

1. Install Adaptive Server ADO.NET Data Provider using the installation program on the end user machine.
2. If the .NET Framework SDK 1.1 is not installed on this machine, then manually deploy the provider assembly into the GAC.
3. Copy your application-specific files such as *exe*, and *dlls*, to the system in the application-specific folder.

❖ **Using the GAC to deploy an application**

1. Copy the *dll* files that make up Adaptive Server ADO.NET Data Provider on the target machine in a directory such as *C:\Sybase\DataAccess\ADONET\dll*.
2. Add this directory to the system path.
3. Deploy the provider assembly into the GAC. See "Deploying Adaptive Server ADO.NET Data Provider assembly into the global assembly cache" on page 3 for details.
4. Copy your application-specific files (such as *exe*, and *dlls*), to the system in the application-specific folder.
5. Execute the application.

❖ **Deploying an application independent of the GAC**

1. On the target system, copy the *dll* files that make up Adaptive Server ADO.NET Data Provider in the application-specific folder, in addition to the application-specific files, such as *exe*, and *dlls*.
2. Execute the application.

Updating to a newer version of ADO.NET Data Provider

Updates to Adaptive Server ADO.NET Data Provider are delivered either through an EBF/ESD or maintenance releases. This section covers issues about updating to the newer version of Data Provider. For more information about Microsoft .NET concepts that pertain to updating, read the *.NET Framework Deployment Guide* and *.NET Framework Developer's Guide* found on the Microsoft Developer Network at http://msdn.microsoft.com:

To migrate your applications to the new version of Adaptive Server Data Provider, perform one of the following:

- Create an application configuration file to redirect your applications to use the new version of Adaptive Server ADO.NET Data Provider. See "Using application configuration files" on page 6.
- Rebuild and redeploy your application against the new version of Adaptive Server ADO.NET Data Provider. Sybase recommends that you choose this step.

Redirecting the Common Language Runtime

The .NET Common Language Runtime (CLR) locates and binds references to assemblies, such as Data Provider, in the application program when it executes. By default, the CLR attempts to bind references to the exact version of the assembly that the application was built with. Consequently, your applications will not automatically use an updated version of the assembly just because you have deployed it; you must rebuild the application against this new version of the assembly, or the CLR needs to be directed by a configuration file to use the newer version of the assembly.

Usually, EBF/ESD releases of Data Provider for the same release level (major and minor) are binary-compatible with the previous release. It is possible for such updates to preclude rebuilding your application. If you do not want to rebuild and redeploy your applications for each update to Data Provider, you can use application configuration or Publisher policy files. Sybase usually includes a Publisher policy file in the ESD/EBF releases with the appropriate redirection. See the ESD/EDF documentation for information on backward-compatibility issues.

Using application configuration files

You can use an application configuration file to direct the CLR to load a version of an assembly different than the one stored in the calling application's manifest.

This example shows how an application can direct the CLR to use Data Provider build 1.0.159 for an application built against any prior 1.0.x build of Data Provider:

```
<configuration>
  <runtime>
    <assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
      <dependentAssembly>
        <assemblyIdentity name="Sybase.Data.AseClient"
                          publicKeyToken="26e0f1529304f4a7"
                          culture="neutral" />
        <bindingRedirect oldVersion="1.0.0.0-1.0.158.65535"
                         newVersion="1.0.159.0"/>
      </dependentAssembly>
    </assemblyBinding>
  </runtime>
</configuration>
```

Refer to the MSDN Library at http://msdn2.microsoft.com/en-us/default.aspx for complete configuration file schema information.

Note Each application needs its own separate configuration file.

Using Publisher policy files

A Publisher policy file can be distributed by the publisher of the assembly along with the update fix to a shared assembly. This file directs all references from an older assembly version to the newly installed version. Unlike the application configuration files, Publisher policy files need to be deployed in the global assembly cache (GAC) to become functional.

The settings in the Publisher policy file override the version information that comes from the application or application configuration file. However, specific applications can be set up to ignore the Publisher policy file by enforcing "safe mode." Refer to the MSDN Library for information on setting applications to use safe mode.

Updates to Adaptive Server ADO.NET Data Provider usually include a Publisher policy file that redirects applications to the latest installed version of the Data Provider assembly. If the installation program detects that .NET Framework SDK 1.1 is installed on the system, it also deploys the new provider assembly and the Publisher policy file in the GAC.

Deploying updates to the Data Provider

The following sections contain issues related to deploying updates to the Data Provider.

Deploying Data Provider into the GAC

If the updated Data Provider assembly and the policy assembly are deployed into the GAC, all of the applications on this system automatically begin to use the updated Provider.

Excluding specific applications from using the updated Data Provider

If you want to exclude a specific application from using this updated Data Provider, you can set up an application configuration file for this application that forces safe mode to override the Publisher policy file.

Core file locations

Data Provider consists of two core files: *Sybase.Data.AseClient.dll* and *sybdrvado115.dll*. Multiple versions of Sybase.Data.AseClient.dll can be installed in the GAC. However, the *sybdrvado115.dll* is not installed in the GAC and is located at runtime using the system PATH. This file is installed in the Data Provider installation directory. Sybase may change the name or version string for this DLL in upgrade releases. For example, in the 1.0 release this file was called *aseado.dll*, and in the 1.1 release it is called *sybdrvado115.dll*. When such an update is installed, do not delete the older version of this file unless Data Provider versions using this file are removed from the GAC. Otherwise, applications attempting to use the older version of the provider will fail.

Deploying Data Provider when it is not in the GAC

If the Data Provider assembly is not installed in the GAC on this machine, the files that make up Data Provider need to be copied into the application folder.

To make the application use an updated version of Data Provider, you can do one of the following:

- Create an application configuration file with the appropriate redirect.
- Rebuild the application against the new Data Provider.
- Deploy only the Publisher policy file into the GAC. Doing this causes all references to Data Provider on this machine to use the redirect in the Publisher policy file unless specifically excluded by an application.

Running the sample projects

Three sample projects are included with Adaptive Server ADO.NET Data Provider:

- **Simple –** a sample program that demonstrates how to connect to a database, execute a query, and read resultsets returned.
- **TableViewer –** a sample program that demonstrates how the AseDataAdapter object can be used to bind results to a DataGrid control.
- **Advanced –** a sample program that demonstrates how to call stored procedures with input, output, and inout parameters; read the stored procedure return value; use two supported mechanisms to pass parameters; and use the tracing feature of the provider.

For tutorials explaining the Simple and Table Viewer samples, see Chapter 2, "Using the Sample Applications."

The pubs2 database, which is required to run Adaptive Server ADO.NET Data Provider samples, is not installed by default with Adaptive Server. See the *Adaptive Server Enterprise Installation Guide* for instructions on how to install the pubs2 database.

CHAPTER 2

Using the Sample Applications

This chapter explains how to use the sample projects included with Adaptive Server ADO.NET Data Provider.

Note To run the sample programs, you need access to an Adaptive Server with the pubs2 sample database installed; also, you need either the Visual Studio .NET 2003 or the .NET Framework 1.1 to install on the system.

The sample programs are located in the following directories in your Adaptive Server ADO.NET Data Provider installation directory:

- *Samples\CSharp*, which contains the three samples written in the C# programming language
- *Samples\VB.NET*, which contains the three samples written in the Visual Basic .NET programming language

The default installation directory is *C:\Sybase\DataAccess\ADONET\dll*.

Tutorial: Using the Simple code sample

The Simple project illustrates the following features:

- Connecting to a database
- Executing a query using the AseCommand object
- Using the AseDataReader object
- Basic error handling

For more information about how the sample works, see “Understanding the Simple sample project” on page 13.

❖ **Running the Simple code sample in Visual Studio .NET**

1. Start Visual Studio .NET.
2. Choose File | Open | Project.
3. Browse to the sample project:

   For C#, browse to *<install dir>\Samples\CSharp\Simple* and open *Simple.csproj*.

   For Visual Basic .NET, browse to *<install dir>\Samples\VB.NET\Simple* and open *Simple.vbproj*.
4. If you have installed Adaptive Server ADO.NET Data Provider using the installation program, go directly to step 7.
5. If you have not used the installation program, then you need to correct references to the Adaptive Server ADO.NET Data Provider in the project. To do this, delete the existing reference first:
   a. In the Solution Explorer window, verify that the Simple project is expanded.
   b. Expand the References folder.
   c. Right-click *Sybase.AseClient.Data.dll* and select Remove.
6. Add a reference to Adaptive Server ADO.NET Data Provider Assembly.

   For instructions, see “Adding a reference to the Data Provider assembly” on page 29.
7. To run the Simple sample, choose Debug | Start Without Debugging, or press Ctrl+F5.

   The AseSample dialog box appears.
8. In the AseSample dialog box, provide connection information for an Adaptive Server with the sample pubs2 database and then click Connect.

   The application connects to the sample pubs2 database and puts the last name of each author in the dialog box.
9. Click X in the upper right corner of the window to terminate the application and disconnect from the pubs2 database.

You have now run the application. The next section describes the application code.

❖ **Running the Simple sample project without Visual Studio**

1. From a DOS prompt, go to the appropriate sample directory under *<install dir>\Samples*.
2. Add the directory with .NET Framework 1.1 binaries to your system path.
3. Verify that the *dll* directory under Adaptive Server ADO.NET Data Provider installation directory is included in the system path and the LIB environment variable.
4. Compile the sample program using the supplied build script called *build.bat*.
5. To run the program, enter:

   ```
   simple.exe
   ```

   The AseSample dialog box appears.
6. In the AseSample dialog box, provide connection information for an Adaptive Server with the sample pubs2 database and then click Connect.

   The application connects to the sample pubs2 database and puts the last name of each author in the dialog box.
7. Click the X in the upper right corner of the window to terminate the application and disconnect from the pubs2 database.

Understanding the Simple sample project

This section illustrates some key features of Adaptive Server ADO.NET Data Provider by walking through some of the code from the Simple code sample, which uses the Adaptive Server sample database, pubs2. See the *Adaptive Server Enterprise Installation Guide* to find out how to install the pubs2 database.

This section describes portions of the code. To see all of the code, open the sample project:

For C#:

```
<install dir>\Samples\CSharp\Simple\Simple.csproj
```

For Visual Basic .NET:

```
<install dir>\Samples\VB.NET\Simple\Simple.vbproj
```

Declaring imports

At the beginning of the program, it declares the import statement to import Adaptive Server ADO.NET Data Provider information:

For C#:

```
using Sybase.Data.AseClient;
```

For Visual Basic .NET:

```
Imports Sybase.Data.AseClient
```

Connecting to the database

The btnConnect_Click method declares and initializes a connection object called new AseConnection:

For C#:

```
AseConnection conn = new AseConnection(
   "Data Source='" + host +
      "';Port='" + port +
      "';UID='" + user +
      "';PWD='" + pass +
      "';Database='pubs2';" );
```

For Visual Basic .NET:

```
Dim conn As New AseConnection( _
     "Data Source='" + host + _
     "';Port='" + port + _
     "';UID='" + user + _
     "';PWD='" + pass + _
     "';Database='pubs2';")
```

The AseConnection object uses the connection string to connect to the sample database.

For C#:

```
conn.Open();
```

For Visual Basic .NET:

```
conn.Open()
```

For more information about the AseConnection object, see “AseConnection class” on page 114.

Executing a query

The following code uses the Command object (AseCommand) to define and execute a SQL statement. Then, it returns the DataReader object (AseDataReader):

For C#:

```
AseCommand cmd = new AseCommand( "select au_lname from
     authors", conn );
AseDataReader reader = cmd.ExecuteReader();
```

For Visual Basic .NET:

```
Dim cmd As New AseCommand( _
     "select au_lname from authors", conn)
Dim reader As AseDataReader  = cmd.ExecuteReader()
```

For more information about the Command object, see “AseCommand class” on page 101.

Displaying the results

The following code loops through the rows held in the AseDataReader object and adds them to the ListBox control. The DataReader uses `GetString( 0 )` to get the first value from the row.

Each time the Read method is called, the DataReader gets another row back from the result set. A new item is added to the ListBox for each row that is read:

For C#:

```
listAuthors.BeginUpdate();
while( reader.Read() ) {
     listAuthors.Items.Add( reader.GetString( 0 ) );
}
listAuthors.EndUpdate();
```

For Visual Basic .NET:

```
listAuthors.BeginUpdate()
While reader.Read()
     listAuthors.Items.Add(reader.GetString(0))
End While
listAuthors.EndUpdate()
```

For more information about the AseDataReader object, see “AseDataReader class” on page 134.

Finishing off

The following code at the end of the method closes the reader and connection objects:

For C#:

```
reader.Close();
conn.Close();
```

For Visual Basic .NET:

```
reader.Close()
conn.Close()
```

Error handling

Any errors that occur during execution and that originate with Adaptive Server ADO.NET Data Provider objects are displayed in a message box. The following code catches the error and displays its message:

For C#:

```
catch( AseException ex ) {
     MessageBox.Show( ex.Message );
}
```

For Visual Basic .NET:

```
Catch ex As AseException
     MessageBox.Show(ex.Message)
End Try
```

For more information about the AseException object, see "AseException class" on page 153.

Tutorial: Using the Table Viewer code sample

This tutorial is based on the Table Viewer project that is included with Adaptive Server ADO.NET Data Provider. The complete application can be found in your Adaptive Server ADO.NET Data Provider installation directory.

For C#:

```
 <install
dir>\Samples\CSharp\TableViewer\TableViewer.csproj
```

For Visual Basic .NET:

```
<install
dir>\Samples\VB.NET\TableViewer\TableViewer.vbproj
```

The Table Viewer project is more complex than the Simple project. It illustrates the following features:

- Connecting to a database
- Working with the AseDataAdapter object
- More advanced error handling and result checking

For more information about how the sample works, see "Understanding the Table Viewer sample project" on page 18.

❖ **Running the Table Viewer code sample in Visual Studio .NET**

1 Start Visual Studio .NET.

2 Choose File | Open | Project.

3. Browse to the *Samples* directory in your Adaptive Server ADO.NET Data Provider installation directory. Go to the *CSharp* or *VB.NET* directory and open the Table viewer project.
4. If you have installed Adaptive Server ADO.NET Data Provider using the installation program, go directly to step 7.
5. If you have not used the installation program, then you need to correct references to Adaptive Server ADO.NET Data Provider in the project. To do this, delete the existing reference first:
   a. In the Solution Explorer window, verify that the Simple project is expanded.
   b. Expand the References folder.
   c. Right-click *Sybase.AseClient.Data.dll* and select Remove.
6. Add a reference to the Adaptive Server ADO.NET Data Provider Assembly.

   For instructions, see "Adding a reference to the Data Provider assembly" on page 29.
7. To run the TableViewer sample, choose Debug | Start Without Debugging or press Ctrl+F5.
8. In the Table Viewer dialog box, supply connection information to an Adaptive Server with pubs2 sample database installed. Click Connect.

   The application connects to the Adaptive Server pubs2 sample database.
9. In the Table Viewer dialog box, click Execute.

   The application retrieves the data from the authors table in the sample database and puts the query results in the Results DataList.

   You can also execute other SQL statements from this application. Enter a SQL statement in the SQL Statement pane and click Execute.
10. Click the X in the upper right-hand corner of the window to terminate the application and disconnect from the sample database.

❖ **Running the Table viewer sample project without Visual Studio**

1. Open a DOS prompt and go to the appropriate sample directory under *<install directory>\Samples*.
2. Add the directory with .NET Framework 1.1 binaries to your system path.

3 Verify that the *dll* directory under Adaptive Server ADO.NET Data Provider installation directory is included in the system path and the LIB environment variable.

4 Compile the sample program using the supplied build script *build.bat*.

5 To run the program, enter:

```
tableviewer.exe
```

6 In the Table Viewer dialog box, supply connection information to an Adaptive Server with pubs2 sample database installed.

   Click Connect.

   The application connects to the Adaptive Server pubs2 sample database.

7 In the Table Viewer dialog box, click Execute.

   The application retrieves the data from the authors table in the sample database and puts the query results in the Results DataList.

   You can also execute other SQL statements from this application: Enter a SQL statement in the SQL Statement pane, and then click Execute.

8 Click the X in the upper right-hand corner of the window to terminate the application and disconnect from the sample database.

You have now run the application. The next section describes the application code.

Understanding the Table Viewer sample project

This section illustrates some key features of Adaptive Server ADO.NET Data Provider by walking through some of the code from the Table Viewer code sample. The Table Viewer project uses the Adaptive Server sample database, pubs2, which can be installed from the scripts located in the Adaptive Server installation directory.

In this section, the code is described a few lines at a time. To see all the code, open the sample project in Adaptive Server installation directory.

For C#:

```
<install dir> \Samples\CSharp\TableViewer\
TableViewer.csproj
```

For Visual Basic .NET:

```
<install dir>\Samples\VB.NET\TableViewer
\TableViewer.vbproj
```

Declaring imports

At the beginning of the program, it declares the import statement to import the Adaptive Server ADO.NET Data Provider information:

For C#:

```
using Sybase.Data.AseClient;
```

For Visual Basic .NET:

```
Imports Sybase.Data.AseClient
```

Declaring an instance variable

Use the AseConnection class to declare an instance variable of type AseConnection. This connection is used for the initial connection to the database, as well as when you click Execute to retrieve the result set from the database:

For C#:

```
private AseConnection
     _conn;
```

For Visual Basic .NET:

```
Private _conn As AseConnection
```

For more information, see “AseConnection constructors” on page 114.

Connecting to the database

The following code provides a default value for the connection string that appears in the Connection String field by default:

For C#:

```
txtConnectString.Text = "Data Source='" +
   System.Net.Dns.GetHostName() +
   "';Port='5000';UID='sa';PWD='';Database='pubs2';";
```

For Visual Basic .NET:

```
txtConnectString.Text = "Data Source='" + _
   System.Net.Dns.GetHostName() + _
   "';Port='5000';UID='sa';PWD='';Database='pubs2';"
```

The Connection object later uses the connection string to connect to the sample database:

For C#:

```
_conn = new AseConnection( txtConnectString.Text );
_conn.Open();
```

For Visual Basic .NET:

```
_conn = New AseConnection(txtConnectString.Text)
_conn.Open()
```

For more information, see “AseConnection class” on page 114.

Defining a query

The following code defines the default query that appears in the SQL Statement field:

For C#:

```
this.txtSQLStatement.Text = "SELECT * FROM authors";
```

For Visual Basic .NET:

```
Me.txtSQLStatement.Text = "SELECT * FROM authors"
```

Displaying the results

Before the results are fetched, the application verifies whether the Connection object has been initialized. If it has, it ensures that the connection state is open:

For C#:

```
if( _conn == null || _conn.State != ConnectionState.Open )
{
     MessageBox.Show( "Connect to a database first.", "Not connected" );
     return;
}
```

For Visual Basic .NET:

```
If (_conn Is Nothing) OrElse (_conn.State <> ConnectionState.Open) Then
     MessageBox.Show("Connect to a database first.", "Not connected")
     Return
End If
```

When you are connected to the database, the following code uses the DataAdapter object (AseDataAdapter) to execute the SQL statement. A new DataSet object is created and filled with the results from the DataAdapter object. Finally, the contents of the DataSet are bound to the DataGrid control on the window.

For C#:

```
using(AseCommand cmd = new AseCommand( txtSQLStatement.Text.Trim(), _conn ))
{
     using(AseDataAdapter   da = new AseDataAdapter(cmd))
     {
          DataSet ds = new DataSet();
          da.Fill(ds, "Table");

          dgResults.DataSource = ds.Tables["Table"];
     }
```

```
}
```

For Visual Basic .NET:

```
Dim cmd As New AseCommand( _
   txtSQLStatement.Text.Trim(), _conn)
Dim da As New AseDataAdapter(cmd)
Dim ds As New DataSet
da.Fill(ds, "Table")
dgResults.DataSource = ds.Tables("Table")
```

Because a global variable is used to declare the connection, the connection that was opened earlier is reused to execute the SQL statement.

For more information about the DataAdapter object, see “AseDataAdapter class” on page 126.

Error handling

If an error occurs when the application attempts to connect to the database, the following code catches the error and displays its message:

For C#:

```
catch( AseException ex )
{
   MessageBox.Show( ex.Source + " : " + ex.Message +
      " ("+ ex.ToString() + ")",
      "Failed to connect" );
}
```

For Visual Basic .NET:

```
Catch ex As AseException
   MessageBox.Show(ex.Source + " : " + ex.Message + _
   "(" + ex.ToString() + ")" + _
   "Failed to connect")
End Try
```

Tutorial: Using the Advanced code sample

This tutorial is based on the Advanced project that is included with Adaptive Server ADO.NET Data Provider. The complete application can be found in your Adaptive Server ADO.NET Data Provider installation directory:

For C#:

```
<install dir>\Samples\CSharp\Advanced\Advanced.csproj
```

For Visual Basic .NET:

```
<install dir>\Samples\VB.NET\Advanced\Advanced.vbproj
```

The Advanced project illustrates the following features:

- Connecting to a database
- Using the trace event feature to trace ADO.NET calls made to Adaptive Server ADO.NET Data Provider

  You can use the trace event feature to log all the ADO.NET calls you make to troubleshooting and gathering more information for Sybase Technical Support.
- Using named parameters ("@param")
- Using parameter markers ("?"), for example: `{? = call sp_hello(?, ?, ?)}`
- Calling stored procedures using input, input/output, output parameters, and return values. There are two ways you can call stored procedures in Adaptive Server:
  - Using the name of the stored procedure as CommandText and setting AseCommand.CommandType to CommandType.StoredProcedure.
  - Using call syntax, which is compatible with ODBC and JDBC programs.

❖ **Running the Advanced code sample in Visual Studio .NET**

1. Start Visual Studio .NET.
2. Choose File | Open | Project.
3. Browse to the *Samples* directory in your Adaptive Server ADO.NET Data Provider installation directory. Go to the *CSharp* or *VB.NET* directory and open the Advanced project.
4. If you have installed Adaptive Server ADO.NET Data Provider using the installation program, go directly to step 7.
5. If you have not used the installation program, you need to correct references to the Adaptive Server ADO.NET Data Provider in the project. To do this, delete the existing reference first:
   a. In the Solution Explorer window, verify that the Simple project is expanded.
   b. Expand the References folder.

c Right-click *Sybase.AseClient.Data.dll* and select Remove.

6 Add a reference to the Adaptive Server ADO.NET Data Provider Assembly.

7 Choose Debug | Start Without Debugging to run the Advanced project.

The Form1 dialog box appears.

8 In the Form1 dialog box, click Connect.

The application connects to the Adaptive Server sample database.

9 In the Form1 dialog box, click Execute.

The application executes the stored procedure and gets back an input-output parameter, output parameter, and a return value.

10 Click the X in the upper right-hand corner of the window to terminate the application and disconnect from the sample database.

You have now run the application. The next section describes the application.

❖ **Running the Advanced sample project without Visual Studio**

1 Open a DOS prompt and go to the appropriate sample directory under the *<install directory>\Samples* directory.

2 Add the directory with .NET Framework 1.1 binaries to your system path.

3 Verify that the *dll* directory under the Adaptive Server ADO.NET Data Provider installation directory, is included in the system path and the LIB environment variable.

4 Compile the sample program using the supplied build script *build.bat.*

5 To run the program, enter:

```
advanced.exe
```

6 The Form1 dialog box appears. Click Connect.

The application connects to the Adaptive Server sample database.

7 In the Form1 dialog box, click Execute.

The application executes the stored procedure and gets back an input-output parameter, output parameter, and a return value.

8 Click the X in the upper right-hand corner of the window to terminate the application and disconnect from the sample database.

You have now run the application. The next section describes the application code.

Understanding the Advanced sample project

This section illustrates some key features of Adaptive Server ADO.NET Data Provider by walking through some of the code from the Advanced code sample. The Advanced project uses the Adaptive Server sample database, pubs2, which can be installed from your Adaptive Server CDs.

In this section, the code is described a few lines at a time. To see all of the code, open the sample project:

For C#:

```
<install dir>\Samples\CSharp\Advanced\Advanced.csproj
```

For Visual Basic .NET:

```
<install dir>\Samples\VB.NET\Advanced\Advanced.vbproj
```

Attaching trace event handlers

The following lines of code attaches your trace event handlers to the AseConnection:

For C#:

```
_conn.TraceEnter += new
     TraceEnterEventHandler(TraceEnter);
_conn.TraceExit += new
     TraceExitEventHandler(TraceExit);
```

For Visual Basic .NET:

```
AddHandler _conn.TraceEnter, AddressOf TraceEnter
AddHandler _conn.TraceExit, AddressOf TraceExit
```

Invoking the stored procedures using named parameters

The method ExecuteCommandUsingNamedParams() invokes the stored procedure by name using named parameters.

For C#:

```
using(AseCommand cmd = new AseCommand("sp_hello", _conn))
{
   cmd.CommandType = CommandType.StoredProcedure;

   AseParameter inParam = new AseParameter("@inParam", AseDbType.VarChar, 32);
   inParam.Direction = ParameterDirection.Input;
   inParam.Value = textBoxInput.Text;
   cmd.Parameters.Add(inParam);
```

```
   AseParameter inoutParam = new AseParameter("@inoutParam",
   AseDbType.VarChar, 64);
   inoutParam.Direction = ParameterDirection.InputOutput;
   inoutParam.Value = textBoxInOut.Text;
   cmd.Parameters.Add(inoutParam);

   AseParameter outParam = new AseParameter("@outParam",
   AseDbType.VarChar, 64);
   outParam.Direction = ParameterDirection.Output;
   cmd.Parameters.Add(outParam);

   AseParameter retValue = new AseParameter("@retValue", AseDbType.Integer);
   retValue.Direction = ParameterDirection.ReturnValue;
   cmd.Parameters.Add(retValue);

   try
   {
      cmd.ExecuteNonQuery();
   }
   catch (AseException ex)
   {
      MessageBox.Show(ex.Source + " : " + ex.Message + " (" + ex.ToString() +
      ")", "Execute Stored Precedure failed.");
   }
}
```

For Visual Basic .NET:

```
Dim cmd As New AseCommand("sp_hello", _conn)
' set command type to stored procedure
cmd.CommandType = CommandType.StoredProcedure

' create the input parameter object and bind it to the command
Dim inParam As New AseParameter("@inParam", AseDbType.VarChar, 32)
inParam.Direction = ParameterDirection.Input
inParam.Value = textBoxInput.Text
cmd.Parameters.Add(inParam)

' create the inout parameter object and bind it to the command
Dim inoutParam As New AseParameter("@inoutParam", AseDbType.VarChar, 64)
inoutParam.Direction = ParameterDirection.InputOutput
inoutParam.Value = textBoxInOut.Text
cmd.Parameters.Add(inoutParam)

' create the output parameter object and bind it to the command
Dim outParam As New AseParameter("@outParam", AseDbType.VarChar, 64)
```

```
outParam.Direction = ParameterDirection.Output
cmd.Parameters.Add(outParam)

' create the return value object and bind it to the command
Dim retValue As New AseParameter("@retValue", AseDbType.Integer)
retValue.Direction = ParameterDirection.ReturnValue
cmd.Parameters.Add(retValue)

' execute the stored procedure
Try
   cmd.ExecuteNonQuery()

Catch ex As AseException
   MessageBox.Show(ex.Source + " : " + ex.Message + " (" + ex.ToString() + ")",
      "Execute Query failed.")
Finally
   ' dispose the command object
   cmd.Dispose()
End Try
```

Invoking the stored procedures using call syntax and parameter markers

The method ExecuteCommandUsingParameterMarkers() invokes the stored procedure using the call syntax and using parameter markers.

For C#:

```
using(AseCommand cmd = new AseCommand("{ ? = call sp_hello(?, ?, ?)}", _conn))
{
   cmd.NamedParameters = false;
   AseParameter retValue = new AseParameter(0, AseDbType.Integer);
   retValue.Direction = ParameterDirection.ReturnValue;
   cmd.Parameters.Add(retValue);

   AseParameter inParam = new AseParameter(1, AseDbType.VarChar, 32);
   inParam.Direction = ParameterDirection.Input;
   inParam.Value = textBoxInput.Text;
   cmd.Parameters.Add(inParam);

   AseParameter inoutParam = new AseParameter(2, AseDbType.VarChar, 64);
   inoutParam.Direction = ParameterDirection.InputOutput;
   inoutParam.Value = textBoxInOut.Text;
   cmd.Parameters.Add(inoutParam);

   AseParameter outParam = new AseParameter(3, AseDbType.VarChar, 64);
   outParam.Direction = ParameterDirection.Output;
   cmd.Parameters.Add(outParam);
   try
   {
```

```
      cmd.ExecuteNonQuery();
   }
   catch (AseException ex)
   {
      MessageBox.Show(ex.Source + " : " + ex.Message + " (" + ex.ToString() +
      ")", "Execute Stored Precedure failed.");
   }
}
```

For Visual Basic .NET:

```
Dim cmd As New AseCommand("{ ? = call sp_hello(?, ?, ?)}", _conn)
' need to notify Named Parameters are not being used (which is the default)
cmd.NamedParameters = False

' create the return value object and bind it to the command
Dim retValue As New AseParameter(0, AseDbType.Integer)
retValue.Direction = ParameterDirection.ReturnValue
cmd.Parameters.Add(retValue)

' create the input parameter object and bind it to the command
Dim inParam As New AseParameter(1, AseDbType.VarChar, 32)
inParam.Direction = ParameterDirection.Input
inParam.Value = textBoxInput.Text
cmd.Parameters.Add(inParam)

' create the inout parameter object and bind it to the command
Dim inoutParam As New AseParameter(2, AseDbType.VarChar, 64)
inoutParam.Direction = ParameterDirection.InputOutput
inoutParam.Value = textBoxInOut.Text
cmd.Parameters.Add(inoutParam)

' create the output parameter object and bind it to the command
Dim outParam As New AseParameter(3, AseDbType.VarChar, 64)
outParam.Direction = ParameterDirection.Output
cmd.Parameters.Add(outParam)

' execute the stored procedure
Try
   cmd.ExecuteNonQuery()

   ' get the output, inout and return values and display them
   textBoxReturn.Text = cmd.Parameters(0).Value
   textBoxReturn.ForeColor = Color.Blue

   textBoxInOut.Text = cmd.Parameters(2).V
```

```
   textBoxOutput.Text = cmd.Parameters(3).Value
   textBoxOutput.ForeColor = Color.Blue
Catch ex As AseException
   MessageBox.Show(ex.Source + " : " + ex.Message + " (" + ex.ToString() + ")",_
   "Execute Query Failed")
Finally
   ' dispose the command object
   cmd.Dispose()
End Try
```

CHAPTER 3

Developing Applications

This chapter describes how to develop and deploy applications with the Adaptive Server ADO.NET Data Provider.

Using Data Provider in a Visual Studio .NET project

After you install Adaptive Server ADO.NET Data Provider, you must make two changes to your Visual Studio .NET project to use it:

- Add a reference to the Adaptive Server ADO.NET Data Provider Assembly.
- Add a line to your source code to reference the Adaptive Server ADO.NET Data Provider classes.

For information about installing and registering Adaptive Server ADO.NET Data Provider, see “Deploying Adaptive Server ADO.NET Data Provider” on page 2.

Adding a reference to the Data Provider assembly

Adding a reference tells Visual Studio .NET which assembly to include to find the code for Adaptive Server ADO.NET Data Provider.

❖ **Adding a reference to Adaptive Server ADO.NET Data Provider in a Visual Studio .NET project**

1. Start Visual Studio .NET and open your project.
2. In the Solution Explorer window, right-click the References folder and choose Add Reference from the pop-up menu.

   The Add Reference dialog box appears.
3. On the .NET tab, scroll through the list of components until you locate the Sybase.Data.AseClient component. Select this component and click Select.
4. Click OK.

   If you do not find the Adaptive Server ADO.NET Data Provider assembly listed in the components, browse to locate *Sybase.Data.AseClient.dll* in the *<install dir>\dll*directory. Select the DLL and click Open. Then, click OK.

   Note The default location is *C:\Sybase\DataAccess\ADONET\dll* for Adaptive Server ADO.NET Data Provider.

   The assembly is added to the References folder in the Solution Explorer window of your project.

Referencing Adaptive Server ADO.NET Data Provider classes

To use Adaptive Server ADO.NET Data Provider, you must also add a line to your source code to reference Adaptive Server ADO.NET Data Provider. You must add a different line for C# than for Visual Basic .NET.

❖ **Referencing the Adaptive Server ADO.NET Data Provider classes in your code**

1. Start Visual Studio .NET and open your project:
   - For C#, add the following line to the list of `using` directives at the beginning of your project:

     ```
     using Sybase.Data.AseClient;
     ```
   - For Visual Basic .NET, add the following line at the beginning of your project before the line `Public Class Form1`:

     ```
     Imports Sybase.Data.AseClient
     ```

This line is not strictly required. However, it allows you to use short forms for the Adaptive Server classes. Without it, you can still use the following in your code:

```
Sybase.Data.AseClient.AseConnection conn = new
Sybase.Data.AseClient.AseConnection();
```

use this line instead of:

```
AseConnection conn = new AseConnection();
```

in your code.

Connecting to a database

Before you can carry out any operations on the data, your application must connect to the database. This section describes how to write code to connect to an Adaptive Server database.

For more information, see “AseConnection class” on page 114 and “ConnectionString property” on page 121.

❖ **Connecting to an Adaptive Server database**

1 Allocate an AseConnection object.

The following code creates an AseConnection object named “conn.”

For C#:

```
AseConnection conn = new AseConnection();
```

For Visual Basic .NET:

```
Dim conn As New AseConnection()
```

You can have more than one connection to a database from your application. Some applications use a single connection to an Adaptive Server database and keep the connection open all the time. To do this, you can declare a global variable for the connection.

For C#:

```
private AseConnection_conn;
```

For Visual Basic .NET:

```
Private _conn As AseConnection
```

For more information, see the code for the Table Viewer sample in <*install dir*>*Samples* and "Understanding the Table Viewer sample project" on page 18.

2 Specify the connection string used to connect to the database:

For C#:

```
AseConnection conn = new AseConnection(
   "Data Source='mango';Port=5000;" +
   "UID='sa';PWD='';" +
   "Database='pubs2';" );
```

where "mango" is the host name where the database server is running.

For Visual Basic .NET:

```
Dim conn As New AseConnection(_
     "Data Source='mango',Port=5000," +_
     "UID='sa';PWD='''';" + _
     "Database='pubs2';")
```

For a complete list of connection parameters, see "AseConnection constructors" on page 114.

3 Open a connection to the database using the following code:

For C#:

```
conn.Open();
```

For Visual Basic .NET:

```
conn.Open()
```

4 Catch connection errors.

Your application should be designed to catch any errors that occur when it attempts to connect to the database. The following code demonstrates how to catch an error and display its message.

For C#:

```
try {
   _conn = new AseConnection(
      txtConnectString.Text );
   _conn.Open();
}
catch( AseException ex ) {
   MessageBox.Show(
      ex.Message,
      "Failed to connect");
```

```
}
```

For Visual Basic .NET:

```
Try
   _conn = New AseConnection(_
      txtConnectionString.Text)
   _conn.Open()
Catch ex As AseException
   MessageBox.Show(_
      ex.Message,_
      "Failed to connect")
End Try
```

Alternately, you can use the ConnectionString property to set the connection string, rather than passing the connection string when the AseConnection object is created.

For C#:

```
AseConnection conn = new AseConnection();
conn.ConnectionString = "Data Source='mango';" +
   "Port=5000;" +
   "UID='sa';" +
   "PWD='';" +
   "Database='pubs2';" ;
```

For Visual Basic .NET:

```
Dim conn As New AseConnection()
conn.ConnectionString = "Data Source='mango';" + _
   "Port=5000;" + _
   "UID='sa';" + _
   "PWD='';" + _
   "Database='pubs2';"
```

where “mango” is the name of the database server.

5 Close the connection to the database. Connections to the database stay open until they are explicitly closed using the conn.Close() method.

Connection pooling

The Adaptive Server Enterprise ADO.NET provider supports connection pooling, which allows your application to reuse existing connections from a pool. To do so, it saves the connection handle to a pool so it can be reused, rather than repeatedly creating a new connection to the database. Connection pooling is turned on by default.

You can also specify the minimum and maximum pool sizes. For example:

```
"Data Source='mango';" +
   "Port=5000;" +
   "UID='sa';" +
   "PWD='';" +
   "Database='pubs2';" +
   "Max Pool Size=50;" +
   "Min Pool Size=5";
```

When your application first attempts to connect to the database, it checks the pool for an existing connection that uses the same connection parameters you have specified. If a matching connection is found, that connection is used. Otherwise, a new connection is used. When you disconnect, the connection is returned to the pool so that it can be reused.

Note If Max Pool Size is specified, Data Provider restricts the maximum number of open connections to this value. The calls to AseConnection.Open() fail with AseException when this limit is reached.

Disabling connection pooling

To disable connection pooling, specify Pooling=False in the connection string.

Checking the connection state

After your application has established a connection to the database, you can check the connection state to verify that the connection is open before you fetch data from the database to update it. If a connection is lost or busy, or if another command is being processed, you can return an appropriate message.

The AseConnection class has a “state property” that checks the state of the connection. Possible state values are Open and Closed.

The following code checks whether the Connection object has been initialized, and if it has, it verifies that the connection is open.

For C#:

```
if( _conn == null || _conn.State !=
   ConnectionState.Open )
   {
      MessageBox.Show( "Connect to a database first",
      "Not connected" );
      return;
   }
```

For Visual Basic .NET:

```
If (_conn Is Nothing) OrElse (_conn.State <>
ConnectionState.Open) Then
     MessageBox.Show("Connection to a database first",
     "Error")
     Return
End If
```

A message is returned if the connection is not open. For more information, see “State property” on page 125.

Character set

The Adaptive Server ADO.NET Data Provider communicates with Adaptive Server using a negotiated character set, which you can configure in the Adaptive Server ADO.NET Data Provider user interface as ClientCharset or Server Default. Server Default is the default setting.

Usage

- When you choose ClientCharset as your character set, also specify a value for the CodePageType property. The valid values are ANSI, the default, and OEM.
- When you choose Other, the value of the charset property is the name of the character set. For example, “charset=utf8” sets the negotiated character set to utf8.
- Avoid using the charset property. .NET strings are unicode and must be converted to multibyte before they are sent to Adaptive Server as char or varchar parameters. Because of this, performance is affected if you use a character set other than the server's default.
- Using an unsupported character set raises this error:

  ```
  [Sybase][driver] Could not load code page for
  requested charset
  ```

  To avoid this error, perform one of the following:

- In your driver's user interface, go to the Advanced tab and select ClientCharset as your character set. If you choose Other, ensure that your specified character set is supported.
- In the connection string, ensure that the charset property specifies a supported character set.

Accessing and manipulating data

With Adaptive Server ADO.NET Data Provider, there are two ways you can access data: using the AseCommand object, or using the AseDataAdapter object.

- **AseCommand object:** The AseCommand object is the recommended way of accessing and manipulating data in .NET because the programmer has more control of connections. However, AseDataAdapter allows you to work offline.

  The AseCommand object allows you to execute SQL statements that retrieve or modify data directly from the database. Using the AseCommand object, you can issue SQL statements and call stored procedures directly against the database.

  Within an AseCommand object, you can use the AseDataReader class to return read-only result sets from a query or stored procedure.

  For more information, see "AseCommand class" on page 101 and "AseDataReader class" on page 134.

- **AseDataAdapter object:** The AseDataAdapter object retrieves the entire result set into a DataSet. A DataSet is a disconnected storage area for data that is retrieved from a database. You can then edit the data in the DataSet, and when you are finished, the AseDataAdapter object updates the database with the changes made to the DataSet. When you use the AseDataAdapter, there is no way to prevent other users from modifying the rows in your DataSet; you need to include logic within your application to resolve any conflicts that may occur.

  For more information, see "Resolving conflicts when using the AseDataAdapter" on page 51.

  For more information about the AseDataAdapter object, see "AseDataAdapter class" on page 126.

Using AseCommand to retrieve and manipulate data

The following sections describe how to retrieve data and how to insert, update, or delete rows using the AseDataReader.

Getting data using the AseCommand object

The AseCommand object allows you to issue a SQL statement or call a stored procedure against an Adaptive Server database. You can issue the following types of commands to retrieve data from the database:

- **ExecuteReader:** Use to issue a command that returns a result set. By default, the Provider does not use cursors. The entire result set is fetched on the client side, and the user can fetch the rows one at a time in forward direction only. If the user turns on the use of cursors by adding the following line to the ConnectString:

  ```
  "Use Cursor=true;"
  ```

  then the Provider does not fetch the whole result set from the database server and instead uses a forward-only, read-only cursor.

  Using cursors can improve performance when you expect your query to return a large resultset, but you do not necessarily expect the client to use the entire resultset.

  In either case, you can loop quickly through the rows of the result set in only one direction.

  For more information, see “ExecuteReader method” on page 104.

- **ExecuteScalar:** Use to issue a command that returns a single value. This can be the first column in the first row of the result set, or a SQL statement that returns an aggregate value such as COUNT or AVG.

  For more information, see “ExecuteScalar method” on page 105.

- **ExecuteXmlReader:** Use to issue a command that returns a result set in an XML format. Generally you use this method in select statements with a FOR XML clause.

  For more information, see “ExecuteXmlReader method” on page 105.

The following instructions use the Simple code sample included with Adaptive Server ADO.NET Data Provider.

For more information about the Simple code sample, see “Understanding the Simple sample project” on page 13.

❖ **Issuing a command that returns a complete result set**

1 Declare and initialize a Connection object:

For C#:

```
AseConnection conn = new AseConnection(connStr);
```

For Visual Basic .NET:

```
Dim conn As New AseConnection(connStr)
```

For C#:

```
try {
     conn.Open();
}
catch (AseException ex)
{
     <error handling>
}
```

For Visual Basic .NET:

```
Try
     conn.Open()
Catch ex As AseExecption
     <error handling>
End Try
```

2 Add a Command object to define and execute a SQL statement:

For C#:

```
AseCommand cmd = new AseCommand(_
   "select au_lname from authors", conn );
```

For Visual Basic .NET:

```
Dim cmd As New AseCommand("select au_lname from
authors", conn)
```

Note When you retrieve data from the database using a stored procedure, and the stored procedure returns both an output parameter value and a result set, then the result set will be reset and you will be unable to reference result set rows as soon as the output parameter value is referenced. In these situations, Sybase recommends that you reference and exhaust all rows in the result set and leave the referencing output parameter value to the end.

For more information, see "Using stored procedures" on page 73 and "AseParameter class" on page 156.

3 Call the ExecuteReader method to return the DataReader object:

For C#:

```
AseDataReader reader = cmd.ExecuteReader();
```

For Visual Basic .NET:

```
Dim reader as AseDataReader = cmd.ExecuteReader()
```

4 Display the results:

For C#:

```
listAuthors.BeginUpdate();
while( reader.Read() ) {
   listAuthors.Items.Add( reader.GetString( 0 ) );
}
listAuthors.EndUpdate();
```

For Visual Basic .NET:

```
 listAuthors.BeginUpdate()
While reader.Read()
     listAuthors.Items.Add(reader.GetString(0))
End While
listAuthors.EndUpdate()
```

5 Close the DataReader and Connection objects:

For C#:

```
reader.Close();
conn.Close();
```

For Visual Basic .NET:

```
reader.close()
conn.close()
```

❖ **Issuing a command that returns only one value**

1 Declare and initialize an AseConnection object:

For C#:

```
AseConnection conn = new AseConnection(
   "Data Source='mango';" +
   "Port=5000;" +
   "UID='sa';" +
   "PWD='';" +
```

```
    "Database='pubs2';" );
```

For Visual Basic .NET:

```
Dim conn As New AseConnection( _
    "Data Source='mango';" + _
    "Port=5000;" + _
    "UID='sa';" + _
    "PWD='';" + _
    "Database='pubs2';")
```

where “mango” is the name of the database server.

2 Open the connection:

For C#:

```
conn.Open();
```

For Visual Basic .NET:

```
conn.Open()
```

3 Add an AseCommand object to define and execute a SQL statement:

For C#:

```
AseCommand cmd = new AseCommand(
  "select count(*) from authors where state = 'CA'",
  conn );
```

For Visual Basic .NET:

```
Dim cmd As New AseCommand(
 "select count(*) from authors where state = 'CA'"_,
 conn );
```

4 Call the ExecuteScalar method to return the object containing the value:

For C#:

```
int count = (int) cmd.ExecuteScalar();
```

For Visual Basic .NET:

```
Dim count As Integer = cmd.ExecuteScalar()
```

5 Close the AseConnection object:

For C#:

```
conn.Close();
```

For Visual Basic .NET:

```
conn.Close()
```

When using the AseDataReader, there are several Get methods available that you can use to return the results in the desired datatype.

For more information, see “AseDataReader class” on page 134.

❖ **Issuing a command that returns an XmlReader object**

1 Declare and initialize a Connection object:

For C#:

```
AseConnection conn = new AseConnection(connStr);
```

For Visual Basic .NET:

```
Dim conn As New AseConnection(connStr)
```

2 Open the connection:

For C#:

```
try {
   conn.Open();
}
catch (AseException ex)
{
   <error handling>
}
```

For Visual Basic .NET:

```
Try
   conn.Open()
Catch ex As AseExecption
   <error handling>
End Try
```

3 Add a Command object to define and execute a SQL statement:

For C#:

```
AseCommand cmd = new AseCommand(
   "select * from authors for xml",
   conn );
```

For Visual Basic .NET:

```
Dim cmd As New AseCommand( _
   "select au_lname from authors for xml", _
   conn
```

4 Call the ExecuteReader method to return the DataReader object:

For C#:

```
XmlReader reader = cmd.ExecuteXmlReader();
```

For Visual Basic .NET:

```
Dim reader as XmlReader = cmd.ExecuteXmlReader()
```

5 Use the XML Result:

For C#:

```
reader.read();
<process xml>
```

For Visual Basic .NET:

```
reader.read()
<process xml>
```

6 Close the DataReader and Connection objects:

For C#:

```
reader.Close();
conn.Close();
```

For Visual Basic .NET:

```
reader.close()
conn.close()
```

Inserting, updating, and deleting rows using the AseCommand object

To perform an Insert, Update, or Delete operation with the AseCommand object, use the ExecuteNonQuery function. The ExecuteNonQuery function issues a command (SQL statement or stored procedure) that does not return a result set.

For more information, see “ExecuteNonQuery method” on page 104.

For information about obtaining primary key values for auto-increment primary keys, see “Obtaining primary key values” on page 63.

If you want to set the isolation level for a command, you must use the AseCommand object as part of an AseTransaction object. When you modify data without an AseTransaction object, Adaptive Server ADO.NET Data Provider operates in autocommit mode, and any changes that you make are applied immediately.

For more information, see “Transaction processing” on page 76.

❖ **Issuing a command that inserts a row**

1 Declare and initialize an AseConnection object:

For C#:

```
AseConnection conn = new AseConnection( c_connStr );
```

For Visual Basic .NET:

```
Dim conn As New AseConnection(c_connStr)
```

2 Open the connection:

For C#:

```
conn.Open();
```

For Visual Basic .NET:

```
conn.Open()
```

3 Add an AseCommand object to define and execute an Insert statement:

For C#:

```
AseCommand insertCmd = new AseCommand(
   "INSERT INTO publishers " +
   " ( pub_id, pub_name, city, state) " +
   " VALUES( @pub_id, @pub_name, @city, @state )",
   conn);
```

For Visual Basic .NET:

```
Dim insertCmd As new AseCommand( _
   "INSERT INTO publishers " + _
   " ( pub_id, pub_name, city, state) " + _
   " VALUES (@pub_id, @pub_name, @city, @state )",  _
   conn )
```

4 Set the parameters for the AseCommand object:

The following code defines parameters for the dept_id and dept_name columns, respectively.

For C#:

```
AseParameter parm = new AseParameter("@pub_id", AseDbType.Char, 4);
insertCmd.Parameters.Add( parm );
parm = new AseParameter("@pub_name", AseDbType.VarChar, 40);
insertCmd.Parameters.Add( parm );
parm = new AseParameter("@city", AseDbType.VarChar, 20);
insertCmd.Parameters.Add( parm );
parm = new AseParameter("@state", AseDbType.Char, 2);
insertCmd.Parameters.Add( parm );
```

For Visual Basic .NET:

```
Dim parm As New AseParameter("@pub_id", AseDbType.Char, 4)
insertCmd.Parameters.Add(parm)
parm = New AseParameter("@pub_name", AseDbType.VarChar, 40)
insertCmd.Parameters.Add(parm)
parm = New AseParameter("@city", AseDbType.VarChar, 20)
insertCmd.Parameters.Add(parm)
parm = New AseParameter("@state", AseDbType.Char, 2)
insertCmd.Parameters.Add(parm)
```

5 Insert the new values and call the ExecuteNonQuery method to apply the changes to the database:

For C#:

```
int recordsAffected = 0;
insertCmd.Parameters[0].Value = "9901";
insertCmd.Parameters[1].Value = "New Publisher";
insertCmd.Parameters[2].Value = "Concord";
insertCmd.Parameters[3].Value = "MA";
recordsAffected = insertCmd.ExecuteNonQuery();
insertCmd.Parameters[0].Value = "9902";
insertCmd.Parameters[1].Value = "My Publisher";
insertCmd.Parameters[2].Value = "Dublin";
insertCmd.Parameters[3].Value = "CA";
recordsAffected = insertCmd.ExecuteNonQuery();
```

For Visual Basic .NET:

```
Dim recordsAffected As Integer
insertCmd.Parameters(0).Value = "9901"
insertCmd.Parameters(1).Value = "New Publisher"
insertCmd.Parameters(2).Value = "Concord"
insertCmd.Parameters(3).Value = "MA"
recordsAffected = insertCmd.ExecuteNonQuery()
insertCmd.Parameters(0).Value = "9902"
insertCmd.Parameters(1).Value = "My Publisher"
insertCmd.Parameters(2).Value = "Dublin"
insertCmd.Parameters(3).Value = "CA"
recordsAffected = insertCmd.ExecuteNonQuery()
```

Note You can use an Insert, Update, or Delete statement with the ExecuteNonQuery method.

6 Display the results and bind them to the grid on the window:

For C#:

```
AseCommand selectCmd = new AseCommand("SELECT * FROM publishers", conn );
```

```
AseDataReader dr = selectCmd.ExecuteReader();
dataGrid.DataSource = dr;
```

For Visual Basic .NET:

```
Dim selectCmd As New AseCommand("SELECT * FROM publishers", conn)
Dim dr As AseDataReader = selectCmd.ExecuteReader()
DataGrid.DataSource = dr
```

7 Close the AseDataReader and AseConnection objects:

For C#:

```
dr.Close();
conn.Close();
```

For Visual Basic .NET:

```
dr.Close()
conn.Close()
```

❖ **Issuing a command that updates a row**

1 Declare and initialize an AseConnection object:

For C#:

```
AseConnection conn = new AseConnection(
   c_connStr );
```

For Visual Basic .NET:

```
Dim conn As New AseConnection(c_connStr)
```

2 Open the connection:

For C#:

```
conn.Open();
```

For Visual Basic .NET:

```
conn.Open()
```

3 Add an AseCommand object to define and execute an update statement:

For C#:

```
AseCommand updateCmd = new AseCommand(
   "UPDATE publishers " +
   "SET pub_name = 'My Publisher' " +
   "WHERE pub_id='9901'",
   conn );
```

For Visual Basic .NET:

```
Dim updateCmd As New AseCommand( _
   "UPDATE publishers " + _
   "SET pub_name = 'My Publisher' " + _
   "WHERE pub_id='9901'", _
   conn )
```

For more information, see “Using stored procedures” on page 73 and “AseParameter class” on page 156.

4 Call the ExecuteNonQuery method to apply the changes to the database:

For C#:

```
int recordsAffected = updateCmd.ExecuteNonQuery();
```

For Visual Basic .NET:

```
Dim recordsAffected As Integer =
   updateCmd.ExecuteNonQuery()
```

5 Display the results and bind them to the grid on the window:

For C#:

```
AseCommand selectCmd = new AseCommand(
   "SELECT * FROM publishers", conn );
AseDataReader dr = selectCmd.ExecuteReader();
dataGrid.DataSource = dr;
```

For Visual Basic .NET:

```
Dim selectCmd As New AseCommand(_
   "SELECT * FROM publishers", conn)
Dim dr As AseDataReader = selectCmd.ExecuteReader()
DataGrid.DataSource = dr
```

6 Close the AseDataReader and AseConnection objects:

For C#:

```
dr.Close();
conn.Close();
```

For Visual Basic .NET:

```
dr.Close()
conn.Close()
```

❖ **Issuing a command that deletes a row**

1 Declare and initialize an AseConnection object:

For C#:

```
AseConnection conn = new AseConnection(c_connStr);
```

For Visual Basic .NET:

```
Dim conn As New AseConnection(c_connStr)
```

2 Open the connection:

For C#:

```
conn.Open();
```

For Visual Basic .NET:

```
conn.Open()
```

3 Create an AseCommand object to define and execute a Delete statement:

For C#:

```
AseCommand updateCmd = new AseCommand
   "DELETE FROM publishers " +
   " WHERE (pub_id > '9900')",
   conn );
```

For Visual Basic .NET:

```
Dim updateCmd As New AseCommand(_
"DELETE FROM publishers " + _
"WHERE (pub_id > '9900')", _
conn )
```

4 Call the ExecuteNonQuery method to apply the changes to the database:

For C#:

```
int recordsAffected = deleteCmd.ExecuteNonQuery();
```

For Visual Basic .NET:

```
Dim recordsAffected As Integer =
   updateCmd.ExecuteNonQuery()
```

5 Close the AseConnection object:

For C#:

```
conn.Close();
```

For Visual Basic .NET:

```
dr.Close()
conn.Close()
```

Obtaining DataReader schema information

You can obtain schema information about columns in the result set.

If you are using the AseDataReader, you can use the GetSchemaTable method to obtain information about the result set. The GetSchemaTable method returns the standard .NET DataTable object, which provides information about all the columns in the result set, including column properties.

For more information about the GetSchemaTable method, see "GetSchemaTable method" on page 142.

❖ **Obtaining information about a result set using the *GetSchemaTable* method**

1 Declare and initialize a connection object:

For C#:

```
AseConnection conn = new AseConnection(
   c_connStr );
```

For Visual Basic .NET:

```
Dim conn As New AseConnection( _
   c_connStr )
```

2 Open the connection:

For C#:

```
conn.Open();
```

For Visual Basic .NET:

```
conn.Open()
```

3 Create an AseCommand object with the Select statement you want to use. The schema is returned for the result set of this query:

For C#:

```
AseCommand cmd = new AseCommand(
   "SELECT * FROM authors", conn );
```

For Visual Basic .NET:

```
Dim cmd As New AseCommand( _
   "SELECT * FROM authors", conn )
```

4 Create an AseDataReader object and execute the Command object you created:

For C#:

```
AseDataReader dr = cmd.ExecuteReader();
```

For Visual Basic .NET:

```
Dim dr As AseDataReader = cmd.ExecuteReader()
```

5 Fill the DataTable with the schema from the data source:

For C#:

```
DataTable
schema = dr.GetSchemaTable();
```

For Visual Basic .NET:

```
Dim schema As DataTable = _
   dr.GetSchemaTable()
```

6 Close the AseDataReader and AseConnection objects:

For C#:

```
dr.Close();
conn.Close();
```

For Visual Basic .NET

```
dr.Close()
conn.Close()
```

7 Bind the DataTable to the grid on the window:

For C#:

```
dataGrid.DataSource = schema;
```

For Visual Basic .NET:

```
dataGrid.DataSource = schema
```

Using AseDataAdapter to access and manipulate data

The following sections describe how to retrieve data and how to insert, update, or delete rows using the AseDataAdapter.

Getting data using the AseDataAdapter object

The AseDataAdapter allows you to view the entire result set by using the Fill method to fill a DataSet with the results from a query by binding the DataSet to the display grid.

Using the AseDataAdapter, you can pass any string (SQL statement or stored procedure) that returns a result set. When you use the AseDataAdapter, by default all the rows are fetched in one operation. If you want the Provider to use cursors, set the property 'use cursor = true' in your connect string. In that case, a forward-only, read-only cursor is used and after all the rows have been read, the cursor is automatically closed by the Provider. The AseDataAdapter allows you to make changes to the DataSet. When your changes are complete, you must reconnect to the database to apply the changes.

You can use the AseDataAdapter object to retrieve a result set that is based on a join.

For more information about the AseDataAdapter, see “AseDataAdapter class” on page 126.

AseDataAdapter example

The following example shows how to fill a DataSet using the AseDataAdapter.

❖ **Retrieving data using the AseDataAdapter object**

1 Connect to the database.

2 Create a new DataSet. In this case, the DataSet is called “Results.”

For C#:

```
DataSet ds =new DataSet ();
```

For Visual Basic .NET:

```
Dim ds As New DataSet()
```

3 Create a new AseDataAdapter object to execute a SQL statement and fill the DataSet called “Results”:

For C#:

```
AseDataAdapter da=new
   AseDataAdapter(txtSQLStatement.Text, _conn);
da.Fill(ds, "Results"),
```

For Visual Basic .NET:

```
Dim da As New
   AseDataAdapter(txtSQLStatement.Text, conn)
da.Fill(ds, "Results")
```

4 Bind the DataSet to the grid on the window:

For C#:

```
dgResults.DataSource = ds.Tables["Results"],
```

For Visual Basic .NET:

```
dgResults.DataSource = ds.Tables("Results")
```

Inserting, updating, and deleting rows using the AseDataAdapter object

The AseDataAdapter object retrieves the result set into a DataSet, which is a collection of tables and the relationships and constraints between those tables. The DataSet is built into the .NET framework and is independent of the Adaptive Server ADO.NET Data Provider used to connect to your database.

When you use the AseDataAdapter, it will open the connection if you are not already connected, fill the DataSet, and close the connection if you had not opened it explicitly. However, when the DataSet is filled, you can modify it while disconnected from the database.

If you do not want to apply your changes to the database right away, you can write the DataSet (including the data and/or the schema) to an XML file using the WriteXml method. Then, you apply the changes at a later time by loading a DataSet with the ReadXml method.

For more information, see the .NET Framework documentation for WriteXml and ReadXml.

When you call the Update method to apply changes from the DataSet to the database, the AseDataAdapter analyzes the changes that have been made and invokes the appropriate commands Insert, Update, or Delete, as necessary.

When you use the DataSet, you can only change (insert, update, or delete) data that is from a single table. You cannot update result sets that are based on joins.

Note Any changes you make to the DataSet are made while you are disconnected. This means that your application does not have locks on these rows in the database. Your application must be designed to resolve any conflicts that can occur when changes from the DataSet are applied to the database if another user changes the data you are modifying before your changes are applied to the database.

Resolving conflicts when using the AseDataAdapter

Some of the conflicts that your application logic should address include:

- *Unique primary keys* – when two users insert new rows into a table, each row must have a unique primary key. For tables with auto-increment primary keys, the values in the DataSet may become out of sync with the values in the data source.

For information about obtaining primary key values for autoincrement primary keys, see "Obtaining primary key values" on page 63.

- *Updates made to the same value* – when two users modify the same value, your application should include logic to determine which value is correct.
- *Schema changes* – when a user modifies the schema of a table you have updated in the DataSet, the update fails when you apply the changes to the database.
- *Data concurrency* – when concurrent applications can see a consistent set of data. The AseDataAdapter does not place a lock on rows that it fetches, so a second user can update a value in the database when you have retrieved the DataSet and are working offline.

Many of these potential problems can be avoided by using the AseCommand, AseDataReader, and AseTransaction objects to apply changes to the database. Sybase recommends the AseTransaction object, because it allows you to set the isolation level for the transaction and it places locks on the rows so that other users cannot modify them.

For more information about using transactions to apply your changes to the database, see "Inserting, updating, and deleting rows using the AseCommand object" on page 42.

To simplify the process of conflict resolution, you can design your insert, update, or delete statement to be a stored procedure call. By including Insert, Update, and Delete statements in stored procedures, you can catch the error if the operation fails. In addition to the statement, you can add error handling logic to the stored procedure so that if the operation fails, the appropriate action is taken, such as recording the error to a log file, or trying the operation again.

❖ Inserting rows into a table using the AseDataAdapter

1. Declare and initialize an AseConnection object:

   For C#:

   ```
   AseConnection conn = new AseConnection(c_connStr);
   ```

   For Visual Basic .NET:

   ```
   Dim conn As New AseConnection( _
      c_connStr )
   ```

2. Open the connection:

   For C#:

   ```
   conn.Open();
   ```

For Visual Basic .NET:

```
conn.Open()
```

3 Create a new AseDataAdapter object:

For C#:

```
AseDataAdapter adapter = new AseDataAdapter();
adapter.MissingMappingAction =
   MissingMappingAction.Passthrough;
adapter.MissingSchemaAction =
   MissingSchemaAction.Add;
```

For Visual Basic .NET:

```
Dim adapter As New AseDataAdapter()
adapter.MissingMappingAction = _
   MissingMappingAction.Passthrough
adapter.MissingSchemaAction = _
   MissingSchemaAction.Add
```

4 Create the necessary AseCommand objects and define any necessary parameters:

The following code creates a Select and an Insert command and defines the parameters for the Insert command:

For C#:

```
adapter.SelectCommand = new AseCommand(
   "SELECT * FROM publishers", conn );
adapter.InsertCommand = new AseCommand(
   "INSERT INTO publishers( pub_id, pub_name, city, state) " +
   "VALUES( @pub_id, @pub_name, @city, @state  )", conn);
adapter.InsertCommand.UpdatedRowSource = UpdateRowSource.None;
AseParameter parm = new AseParameter("@pub_id", AseDbType.Char, 4);
parm.SourceColumn = "pub_id";
parm.SourceVersion = DataRowVersion.Current;
adapter.InsertCommand.Parameters.Add( parm );
parm = new AseParameter("@pub_name", AseDbType.VarChar, 40);
parm.SourceColumn = "pub_name";
parm.SourceVersion = DataRowVersion.Current;
adapter.InsertCommand.Parameters.Add( parm );
parm = new AseParameter("@city", AseDbType.VarChar, 20);
parm.SourceColumn = "city";
parm.SourceVersion = DataRowVersion.Current;
adapter.InsertCommand.Parameters.Add( parm );
parm = new AseParameter("@state", AseDbType.Char, 2);
parm.SourceColumn = "state";
```

```
parm.SourceVersion = DataRowVersion.Current;
adapter.InsertCommand.Parameters.Add( parm );
```

For Visual Basic .NET:

```
adapter.SelectCommand = New AseCommand( _
   "SELECT * FROM publishers", conn )
adapter.InsertCommand = New AseCommand( _
   "INSERT INTO publishers( pub_id, pub_name,  city, state) " + _
   " VALUES( @pub_id, @pub_name, @city, @state  )", conn)
adapter.InsertCommand.UpdatedRowSource = _
   UpdateRowSource.None
Dim parm As New AseParameter("@pub_id", AseDbType.Char, 4)
parm.SourceColumn = "pub_id"
parm.SourceVersion = DataRowVersion.Current
adapter.InsertCommand.Parameters.Add( parm )
parm = New AseParameter("@pub_name", AseDbType.VarChar, 40)
parm.SourceColumn = "pub_name"
parm.SourceVersion = DataRowVersion.Current
adapter.InsertCommand.Parameters.Add( parm )
parm = New AseParameter("@city", AseDbType.VarChar, 20)
parm.SourceColumn = "city"
parm.SourceVersion = DataRowVersion.Current
adapter.InsertCommand.Parameters.Add( parm )
parm = New AseParameter("@state", AseDbType.Char, 2)
parm.SourceColumn = "state"
parm.SourceVersion = DataRowVersion.Current
adapter.InsertCommand.Parameters.Add( parm )
```

5 Fill the DataTable with the results of the Select statement:

For C#:

```
DataTable dataTable = new DataTable( "publishers" );
int rowCount = adapter.Fill( dataTable );
```

For Visual Basic .NET:

```
Dim dataTable As New DataTable( "publishers" )
Dim rowCount As Integer = adapter.Fill( dataTable )
```

6 Insert the new rows into the DataTable and apply the changes to the database:

For C#:

```
DataRow row1 = dataTable.NewRow();
row1[0] = "9901";
row1[1] = "New Publisher";
row1[2] = "Concord";
```

```
row1[3] = "MA";
dataTable.Rows.Add( row1 );
DataRow row2 = dataTable.NewRow();
row2[0] = "9902";
row2[1] = "My Publisher";
row2[2] = "Dublin";
row2[3] = "CA";
dataTable.Rows.Add( row2 );
int recordsAffected = adapter.Update( dataTable );
```

For Visual Basic .NET:

```
Dim row1 As DataRow = dataTable.NewRow()
row1(0) = "9901"
row1(1) = "New Publisher"
row1(2) = "Concord"
row1(3) = "MA"
dataTable.Rows.Add( row1 )
Dim row2 As DataRow = dataTable.NewRow()
row2(0) = "9902"
row2(1) = "My Publisher"
row2(2) = "Dublin"
row2(3) = "CA"
dataTable.Rows.Add( row2 )
Dim recordsAffected As Integer =_
   adapter.Update( dataTable )
```

7 Display the results of the updates:

For C#:

```
dataTable.Clear();
rowCount = adapter.Fill( dataTable );
dataGrid.DataSource = dataTable;
```

For Visual Basic .NET:

```
dataTable.Clear()
rowCount = adapter.Fill( dataTable )
dataGrid.DataSource = dataTable
```

8 Close the connection:

For C#:

```
conn.Close();
```

For Visual Basic .NET:

```
conn.Close()
```

❖ **Updating rows using the AseDataAdapter object**

1 Declare and initialize an AseConnection object:

For C#:

```
AseConnection conn = new AseConnection( c_connStr );
```

For Visual Basic .NET:

```
Dim conn As New AseConnection( _
   c_connStr )
```

2 Open the connection:

For C#:

```
conn.Open();
```

For Visual Basic .NET:

```
conn.Open()
```

3 Create a new AseDataAdapter object:

For C#:

```
AseDataAdapter adapter = new AseDataAdapter();
adapter.MissingMappingAction =
   MissingMappingAction.Passthrough;
adapter.MissingSchemaAction =
   MissingSchemaAction.Add;
```

For Visual Basic .NET:

```
Dim adapter As New AseDataAdapter()
adapter.MissingMappingAction = _
   MissingMappingAction.Passthrough
adapter.MissingSchemaAction = _
   MissingSchemaAction.Add
```

4 Create an AseCommand object and define its parameters.

The following code creates a Select and an Update command and defines the parameters for the Update command:

For C#:

```
adapter.SelectCommand = new AseCommand(
   "SELECT * FROM publishers WHERE pub_id > '9900'",
   conn );
adapter.UpdateCommand = new AseCommand(
   "UPDATE publishers SET pub_name = @pub_name, " +
   "city = @city, state = @state " +
```

```
   "WHERE pub_id = @pub_id", conn );
adapter.UpdateCommand.UpdatedRowSource =
   UpdateRowSource.None;
AseParameter parm = new AseParameter("@pub_id",
   AseDbType.Char, 4);
parm.SourceColumn = "pub_id";
parm.SourceVersion = DataRowVersion.Current;
adapter.UpdateCommand.Parameters.Add( parm );
parm = new AseParameter("@pub_name",
   AseDbType.VarChar, 40);
parm.SourceColumn = "pub_name";
parm.SourceVersion = DataRowVersion.Current;
adapter.UpdateCommand.Parameters.Add( parm );
parm = new AseParameter("@city",
   AseDbType.VarChar, 20);
parm.SourceColumn = "city";
parm.SourceVersion = DataRowVersion.Current;
adapter.UpdateCommand.Parameters.Add( parm );
parm = new AseParameter("@state",
   AseDbType.Char, 2);
parm.SourceColumn = "state";
parm.SourceVersion = DataRowVersion.Current;
adapter.UpdateCommand.Parameters.Add( parm );
```

For Visual Basic .NET:

```
adapter.SelectCommand = New AseCommand( _
  "SELECT * FROM publishers WHERE pub_id > '9900'",_
  conn )
adapter.UpdateCommand = New AseCommand( _
  "UPDATE publishers SET pub_name = @pub_name, " + _
  "city = @city, state = @state " + _
  "WHERE pub_id = @pub_id", conn )
adapter.UpdateCommand.UpdatedRowSource = _
  UpdateRowSource.None
Dim parm As New AseParameter("@pub_id", _
  AseDbType.Char, 4)
parm.SourceColumn = "pub_id"
parm.SourceVersion = DataRowVersion.Current
adapter.UpdateCommand.Parameters.Add( parm )
parm = New AseParameter("@pub_name", _
  AseDbType.VarChar, 40)
parm.SourceColumn = "pub_name"
parm.SourceVersion = DataRowVersion.Current
adapter.UpdateCommand.Parameters.Add( parm )
parm = New AseParameter("@city", _
  AseDbType.VarChar, 20)
```

```
parm.SourceColumn = "city"
parm.SourceVersion = DataRowVersion.Current
adapter.UpdateCommand.Parameters.Add( parm )
parm = New AseParameter("@state", _
  AseDbType.Char, 2)
parm.SourceColumn = "state"
parm.SourceVersion = DataRowVersion.Current
adapter.UpdateCommand.Parameters.Add( parm )
```

5 Fill the DataTable with the results of the Select statement:

For C#:

```
DataTable dataTable = new DataTable( "publishers" );
int rowCount = adapter.Fill( dataTable );
```

For Visual Basic .NET:

```
Dim dataTable As New DataTable( "publishers" )
Dim rowCount As Integer = adapter.Fill( dataTable )
```

6 Update the DataTable with the updated values for the rows, and apply the changes to the database:

For C#:

```
foreach ( DataRow row in dataTable.Rows )
{
   row[1] = ( string ) row[1] + "_Updated";
}
int recordsAffected = adapter.Update( dataTable );
```

For Visual Basic .NET:

```
Dim row as DataRow
For Each row in dataTable.Rows
   row(1) = row(1) + "_Updated"
Next
Dim recordsAffected As Integer = _
adapter.Update( dataTable )
```

7 Bind the results to the grid on the window:

For C#:

```
dataTable.Clear();
adapter.SelectCommand.CommandText =
   "SELECT * FROM publishers";
rowCount = adapter.Fill( dataTable );
dataGrid.DataSource = dataTable;
```

For Visual Basic .NET:

```
dataTable.Clear()
adapter.SelectCommand.CommandText = _
   "SELECT * FROM publishers";
rowCount = adapter.Fill( dataTable )
dataGrid.DataSource = dataTable
```

8 Close the connection:

For C#:

```
conn.Close();
```

For Visual Basic .NET:

```
conn.Close()
```

❖ **Deleting rows from a table using the AseDataAdapter object**

1 Declare and initialize an AseConnection object:

For C#:

```
AseConnection conn = new AseConnection( c_connStr );
```

For Visual Basic .NET:

```
Dim conn As New AseConnection( _
   c_connStr )
```

2 Open the connection:

For C#:

```
conn.Open();
```

For Visual Basic .NET:

```
conn.Open()
```

3 Create an AseDataAdapter object:

For C#:

```
AseDataAdapter adapter = new AseDataAdapter();
adapter.MissingMappingAction =
   MissingMappingAction.Passthrough;
adapter.MissingSchemaAction =
   MissingSchemaAction.AddWithKey;
```

For Visual Basic .NET:

```
Dim adapter As New AseDataAdapter()
adapter.MissingMappingAction = _
   MissingMappingAction.Passthrough
adapter.MissingSchemaAction = _
```

```
MissingSchemaAction.AddWithKey
```

4 Create the required AseCommand objects and define any necessary parameters.

The following code creates a Select and a Delete command and defines the parameters for the Delete command:

For C#:

```
adapter.SelectCommand = new AseCommand(
   "SELECT * FROM publishers WHERE pub_id > '9900'",
   conn );
adapter.DeleteCommand = new AseCommand(
   "DELETE FROM publishers WHERE pub_id = @pub_id",
   conn );
adapter.DeleteCommand.UpdatedRowSource =
   UpdateRowSource.None;
AseParameter parm = new AseParameter("@pub_id",
   AseDbType.Char, 4);
parm.SourceColumn = "pub_id";
parm.SourceVersion = DataRowVersion.Original;
adapter.DeleteCommand.Parameters.Add( parm );
```

For Visual Basic .NET:

```
adapter.SelectCommand = New AseCommand( _
   "SELECT * FROM publishers WHERE pub_id > '9900'", _
   conn )
adapter.DeleteCommand = New AseCommand( _
   "DELETE FROM publishers WHERE pub_id = @pub_id", conn )
adapter.DeleteCommand.UpdatedRowSource = _
   UpdateRowSource.None
Dim parm As New AseParameter("@pub_id", _
   AseDbType.Char, 4)
parm.SourceColumn = "pub_id"
parm.SourceVersion = DataRowVersion.Original
adapter.DeleteCommand.Parameters.Add( parm )
```

5 Fill the DataTable with the results of the Select statement:

For C#:

```
DataTable dataTable = new DataTable( "publishers" );
int rowCount = adapter.Fill( dataTable );
```

For Visual Basic .NET:

```
Dim dataTable As New DataTable( "publishers" )
Dim rowCount As Integer = adapter.Fill( dataTable )
```

6 Modify the DataTable and apply the changes to the database:

For C#:

```
foreach ( DataRow row in dataTable.Rows )
{
   row.Delete();
}
int recordsAffected = adapter.Update( dataTable );
```

For Visual Basic .NET:

```
Dim row as DataRow
For Each row in dataTable.Rows
   row.Delete()
Next
Dim recordsAffected As Integer =_
   adapter.Update( dataTable )
```

7 Bind the results to the grid on the window:

For C#:

```
dataTable.Clear();
rowCount = adapter.Fill( dataTable );
dataGrid.DataSource = dataTable;
```

For Visual Basic .NET:

```
dataTable.Clear()
rowCount = adapter.Fill( dataTable )dataGrid.
DataSource = dataTable
```

8 Close the connection:

For C#:

```
conn.Close();
```

For Visual Basic .NET:

```
conn.Close()
```

Obtaining AseDataAdapter schema information

When using the AseDataAdapter, you can use the FillSchema method to obtain schema information about the result set in the DataSet. The FillSchema method returns the standard .NET DataTable object, which provides the names of all the columns in the result set.

For more information, see “FillSchema method” on page 129.

❖ **Obtaining DataSet schema information using the *FillSchema* method**

1 Declare and initialize an AseConnection object:

For C#:

```
AseConnection   conn = new AseConnection(
   c_connStr );
```

For Visual Basic .NET:

```
Dim conn As New AseConnection( _
   c_connStr )
```

2 Open the connection:

For C#:

```
conn.Open();
```

For Visual Basic .NET:

```
conn.Open()
```

3 Create an AseDataAdapter with the Select statement you want to use. The schema is returned for the result set of this query:

For C#:

```
AseDataAdapter  adapter = new AseDataAdapter(
   "SELECT * FROM employee", conn );
```

For Visual Basic .NET:

```
Dim adapter As New AseDataAdapter( _
   "SELECT * FROM employee", conn )
```

4 Create a new DataTable object, in this case called “Table,” to fill with the schema:

For C#:

```
DataTable dataTable = new DataTable( "Table" );
```

For Visual Basic .NET:

```
Dim dataTable As New DataTable( "Table" )
```

5 Fill the DataTable with the schema from the data source:

For C#:

```
adapter.FillSchema( dataTable, SchemaType.Source );
```

For Visual Basic .NET:

```
adapter.FillSchema( dataTable, SchemaType.Source )
```

6 Close the AseConnection object:

For C#:

```
conn.Close();
```

For Visual Basic .NET:

```
conn.Close()
```

7 Bind the DataSet to the grid on the window:

For C#:

```
dataGrid.DataSource = dataTable;
```

For Visual Basic .NET:

```
dataGrid.DataSource = dataTable
```

Obtaining primary key values

If the table you are updating has an auto-incremented primary key or if the primary key comes from a primary key pool, you can use a stored procedure to obtain values generated by the data source.

When using the AseDataAdapter, this technique can be used to fill the columns in the DataSet with the primary key values generated by the data source. If you use this technique with the AseCommand object, you can either get the key columns from the parameters or reopen the DataReader.

Examples

The following examples use a table called “adodotnet_primarykey” that contains two columns, “id” and “name.” The primary key for the table is “id,” which is a NUMERIC(8) that contains an auto-incremented value; the name column is CHAR(40).

These examples call the following stored procedure to retrieve the auto-incremented primary key value from the database:

```
create procedure sp_adodotnet_primarykey
@p_name char(40),
@p_id int output
as
begin
   insert into adodotnet_primarykey(name)
      VALUES(@p_name)
   select @p_id = @@identity
```

```
END
```

❖ **Inserting a new row with an auto-incremented primary key using the AseCommand object**

1 Connect to the database:

For C#:

```
AseConnection conn = new AseConnection( c_connStr );
conn.Open();
```

For Visual Basic .NET:

```
Dim conn As New AseConnection( _
   c_connStr )
conn.Open()
```

2 Create a new AseCommand object to insert new rows into the DataTable. In the following code, the line `int id1 = ( int ) parmId.Value;` verifies the primary key value of the row:

For C#:

```
AseCommand      cmd = conn.CreateCommand();
cmd.CommandText = "sp_adodotnet_primarykey";
cmd.CommandType = CommandType.StoredProcedure;
AseParameter parmId = new AseParameter(
   "@p_id", AseDbType.Integer);
parmId.Direction = ParameterDirection.Output;
cmd.Parameters.Add( parmId );
AseParameter parmName = new AseParameter(
   "@p_name", AseDbType.Char );
parmName.Direction = ParameterDirection.Input;
cmd.Parameters.Add( parmName );
parmName.Value = "R & D --- Command";
cmd.ExecuteNonQuery();
int id1 = ( int ) parmId.Value;
parmName.Value = "Marketing --- Command";
cmd.ExecuteNonQuery();
int id2 = ( int ) parmId.Value;
parmName.Value = "Sales --- Command";
cmd.ExecuteNonQuery();
int id3 = ( int ) parmId.Value;
parmName.Value = "Shipping --- Command";
cmd.ExecuteNonQuery();
int id4 = ( int ) parmId.Value;
```

For Visual Basic .NET:

```
Dim cmd As AseCommand = conn.CreateCommand()
```

```
cmd.CommandText = "sp_adodotnet_primarykey"
cmd.CommandType = CommandType.StoredProcedure
Dim parmId As New AseParameter("@p_id", _
   AseDbType.Integer)
parmId.Direction = ParameterDirection.Output
cmd.Parameters.Add( parmId )
Dim parmName As New AseParameter("@p_name", _
   AseDbType.Char)
parmName.Direction = ParameterDirection.Input
cmd.Parameters.Add(parmName )

parmName.Value = "R & D --- Command"
cmd.ExecuteNonQuery()
Dim id1 As Integer = parmId.Value
parmName.Value = "Marketing --- Command"
cmd.ExecuteNonQuery()
Dim id2 As Integer = parmId.Value
parmName.Value = "Sales --- Command"
cmd.ExecuteNonQuery()
Dim id3 As Integer = parmId.Value
parmName.Value = "Shipping --- Command"
cmd.ExecuteNonQuery()
dim id4 As Integer = parmId.Value
```

3 Bind the results to the grid on the window, and apply the changes to the database:

For C#:

```
cmd.CommandText = "select * from " +
   "adodotnet_primarykey";
cmd.CommandType = CommandType.Text;
AseDataReader dr = cmd.ExecuteReader();
dataGrid.DataSource = dr;
```

For Visual Basic .NET:

```
cmd.CommandText = "select * from " + _
   "adodotnet_primarykey"
cmd.CommandType = CommandType.Text
Dim dr As AseDataReader = cmd.ExecuteReader()
dataGrid.DataSource = dr
```

4 Close the connection:

For C#:

```
conn.Close();
```

For Visual Basic .NET:

```
conn.Close()
```

❖ **Inserting a new row with an auto-incremented primary key using the AseDataAdapter object**

1 Create a new AseDataAdapter:

For C#:

```
AseConnection conn = new AseConnection(
   c_connStr );
conn.Open();
DataSet dataSet = new DataSet();
AseDataAdapter adapter = new AseDataAdapter();
adapter.MissingMappingAction =
   MissingMappingAction.Passthrough;
adapter.MissingSchemaAction =
   MissingSchemaAction.AddWithKey;
```

For Visual Basic .NET:

```
Dim conn As New AseConnection( _
   c_connStr )
conn.Open()
Dim dataSet As New DataSet()
Dim adapter As New AseDataAdapter()
adapter.MissingMappingAction = _
   MissingMappingAction.Passthrough
adapter.MissingSchemaAction = _
   MissingSchemaAction.AddWithKey
```

2 Fill the data and schema of the DataSet. In the following code, the SelectCommand is called by the AseDataAdapter.Fill method to do this. You can also create the DataSet manually without using the Fill method and SelectCommand if you do not need the existing records:

For C#:

```
adapter.SelectCommand = new AseCommand(
   "select * from adodotnet_primarykey",
   conn );
```

For Visual Basic .NET:

```
adapter.SelectCommand = New AseCommand( _
   "select * from adodotnet_primarykey", conn )
```

3 Create a new AseCommand to obtain the primary key values from the database:

For C#:

```
adapter.InsertCommand = new AseCommand(
   "sp_adodotnet_primarykey", conn );
adapter.InsertCommand.CommandType =
   CommandType.StoredProcedure;
adapter.InsertCommand.UpdatedRowSource =
   UpdateRowSource.OutputParameters;
AseParameter parmId = new AseParameter(
   "@p_id", AseDbType.Integer);
parmId.Direction = ParameterDirection.Output;
parmId.SourceColumn = "id";
parmId.SourceVersion = DataRowVersion.Current;
adapter.InsertCommand.Parameters.Add( parmId );
AseParameter parmName = new AseParameter(
   "@p_name", AseDbType.Char);
parmName.Direction = ParameterDirection.Input;
parmName.SourceColumn = "name";
parmName.SourceVersion = DataRowVersion.Current;
adapter.InsertCommand.Parameters.Add( parmName );
```

For Visual Basic .NET:

```
adapter.InsertCommand = new AseCommand( _
   "sp_adodotnet_primarykey", conn )
adapter.InsertCommand.CommandType = _
   CommandType.StoredProcedure
adapter.InsertCommand.UpdatedRowSource = _
   UpdateRowSource.OutputParameters
Dim parmId As New AseParameter( _
   "@p_id", AseDbType.Integer)
parmId.Direction = ParameterDirection.Output
parmId.SourceColumn = "id"
parmId.SourceVersion = DataRowVersion.Current
adapter.InsertCommand.Parameters.Add( parmId )
Dim parmName As New AseParameter( _
   "@p_name", AseDbType.Char)
parmName.Direction = ParameterDirection.Input
parmName.SourceColumn = "name"
parmName.SourceVersion = DataRowVersion.Current
adapter.InsertCommand.Parameters.Add( parmName )
```

4 Fill the DataSet:

For C#:

```
adapter.Fill( dataSet );
```

For Visual Basic .NET:

```
adapter.Fill( dataSet )
```

5 Insert the new rows into the DataSet:

For C#:

```
DataRow row = dataSet.Tables[0].NewRow();
row[0] = -1;
row[1] = "R & D --- Adapter";
dataSet.Tables[0].Rows.Add( row );
row = dataSet.Tables[0].NewRow();
row[0] = -2;
row[1] = "Marketing --- Adapter";
dataSet.Tables[0].Rows.Add( row );
row = dataSet.Tables[0].NewRow();
row[0] = -3;
row[1] = "Sales --- Adapter";
dataSet.Tables[0].Rows.Add( row );
row = dataSet.Tables[0].NewRow();
row[0] = -4;
row[1] = "Shipping --- Adapter";
dataSet.Tables[0].Rows.Add( row );
```

For Visual Basic .NET:

```
Dim row As DataRow = dataSet.Tables(0).NewRow()
row(0) = -1
row(1) = "R & D --- Adapter"
dataSet.Tables(0).Rows.Add( row )
row = dataSet.Tables(0).NewRow()
row(0) = -2
row(1) = "Marketing --- Adapter"
dataSet.Tables(0).Rows.Add( row )
row = dataSet.Tables(0).NewRow()
row(0) = -3
row(1) = "Sales --- Adapter"
dataSet.Tables(0).Rows.Add( row )
row = dataSet.Tables(0).NewRow()
row(0) = -4
row(1) = "Shipping --- Adapter"
dataSet.Tables(0).Rows.Add( row )
```

6 Apply the changes in the DataSet to the database. When the Update() method is called, the primary key values are changed to the values obtained from the database:

For C#:

```
adapter.Update( dataSet );
dataGrid.DataSource = dataSet.Tables[0];
```

For Visual Basic .NET:

```
adapter.Update( dataSet )
dataGrid.DataSource = dataSet.Tables(0)
```

When you add new rows to the DataTable and call the Update method, the AseDataAdapter calls the InsertCommand and maps the output parameters to the key columns for each new row. The Update method is called only once, but the InsertCommand is called by the Update method as many times as necessary for each new row being added.

7 Close the connection to the database:

For C#:

```
conn.Close();
```

For Visual Basic .NET:

```
conn.Close()
```

Handling BLOBs

When fetching long string values or binary data, there are methods that you can use to fetch the data in pieces. For binary data, use the GetBytes method, and for string data, use the GetChars method. Otherwise, BLOB data is treated in the same manner as any other data you fetch from the database.

For more information, see “GetBytes method” on page 136 and “GetChars method” on page 137.

❖ **Issuing a command that returns a string using the GetChars method**

1 Declare and initialize a Connection object.

2 Open the connection.

3 Add a Command object to define and execute a SQL statement:

For C#:

```
AseCommand cmd = new AseCommand(
   "select au_id, copy from blurbs", conn );
```

For Visual Basic .NET:

```
Dim cmd As New AseCommand( _
   "select au_id, copy from blurbs", conn)
```

4 Call the ExecuteReader method to return the DataReader object:

For C#:

```
AseDataReader reader = cmd.ExecuteReader();
```

For Visual Basic .NET:

```
Dim reader As AseDataReader = cmd.ExecuteReader()
```

The following code reads the two columns from the result set. The first column is a varchar, while the second column is Text. GetChars is used to read 100 characters at a time from the Text column:

For C#:

```
int length = 100;
char[] buf = new char[ length ];
String au_id;
long dataIndex = 0;
long charsRead = 0;
long blobLength = 0;
while( reader.Read() )
{
   au_id = reader.GetString(0);
   do
   {
      charsRead = reader.GetChars(
         1, dataIndex, buf, 0, length);
      dataIndex += length;
      // do something with the chars read
      //.... some code
      //
      // reinitialize char array
      buf = new char[ length ];
   } while ( charsRead == length );
   blobLength = dataIndex + charsRead;
}
```

For Visual Basic .NET:

```
Dim length As Integer = 100
Dim buf(length) As Char
Dim au_id As String
Dim dataIndex As Long = 0
Dim charsRead As Long = 0
Dim blobLength As Long = 0
While reader.Read()
   au_id = reader.GetString(0)
   Do
      charsRead = reader.GetChars( _
```

```
            1, dataIndex, buf, 0, length)
         dataIndex = dataIndex + length
         ' do something with the data read
         '
         '  use code
         '
         ' reinitialize the char array
         ReDim buf(length)
      Loop While (charsRead = length)
      blobLength = dataIndex + charsRead
   End While
```

5 Close the DataReader and Connection objects:

For C#:

```
reader.Close();
conn.Close();
```

For Visual Basic .NET:

```
reader.Close()
conn.Close()
```

Obtaining time values

The .NET Framework does not have a Time structure. If you want to fetch time values from Adaptive Server, you must use the GetDateTime() method. Using this method returns the data as a .NET Framework DateTime object.

❖ **Converting a time value using the *GetDateTime* method**

1 Declare and initialize a connection object:

For C#:

```
AseConnection conn = new AseConnection(
   c_connStr );
```

For Visual Basic .NET:

```
Dim conn As New AseConnection( _
   c_connStr )
```

2 Open the connection:

For C#:

```
conn.Open();
```

For Visual Basic .NET:

```
conn.Open()
```

3 Add a Command object to define and execute a SQL statement:

For C#:

```
AseCommand cmd = new AseCommand(
   "SELECT title_id, title, pubdate FROM titles",
   conn );
```

For Visual Basic .NET:

```
Dim cmd As New AseCommand( _
   "SELECT title_id, title, pubdate FROM titles", _
   conn)
```

4 Call the ExecuteReader method to return the DataReader object:

For C#:

```
AseDataReader reader = cmd.ExecuteReader();
```

For Visual Basic .NET:

```
Dim reader As AseDataReader = cmd.ExecuteReader()
```

The following code uses the GetDateTime method to return the DateTime value:

For C#:

```
while ( reader.Read() )
{
   String tid = reader.GetString(0);
   String title = reader.GetString(1);
   DateTime time = reader.GetDateTime(2);
   // do something with the data
}
```

For Visual Basic .NET:

```
While reader.Read()
   Dim tid As String = reader.GetString(0)
   Dim title As String = reader.GetString(1)
   Dim time As DateTime = reader.GetDateTime(2)
   ' do something with the data....
End While
```

5 Close the DataReader and Connection objects:

For C#:

```
reader.Close();
conn.Close();
```

For Visual Basic .NET:

```
reader.Close()
conn.Close()
```

Using stored procedures

You can use stored procedures with Adaptive Server ADO.NET Data Provider. The ExecuteReader method is used to call stored procedures that return a result set.

Note When you retrieve data from the database using a stored procedure, and the stored procedure returns both an output parameter value and a result set, then the result set will be reset and you will be unable to reference result set rows as soon as the output parameter value is referenced. Sybase recommends that in these situations you reference and exhaust all rows in the result set and leave the referencing output parameter value to the end.

The ExecuteNonQuery method is used to call stored procedures that do not return a result set. The ExecuteScalar method is used to call stored procedures that return only a single value.

If the stored procedure requires parameters you must create equivalent AseParameter objects. If you specify that the CommandType is StoredProcedure, set the CommandText to the name of the stored procedure. For example:

```
sp_producttype
```

For more information about the Parameter object, see “AseParameter class” on page 156.

❖ **Executing a stored procedure**

1 Declare and initialize an AseConnection object:

For C#:

```
AseConnection conn = new AseConnection(
   c_connStr );
```

For Visual Basic .NET:

```
Dim conn As New AseConnection( _
   c_connStr )
```

2 Open the connection:

For C#:

```
conn.Open();
```

For Visual Basic .NET:

```
conn.Open()
```

3 Add an AseCommand object to define and execute a SQL statement. The following code uses the CommandType property to identify the command as a stored procedure:

For C#:

```
AseCommand cmd = new AseCommand(
   "titleid_proc", conn );
cmd.CommandType = CommandType.StoredProcedure;
```

For Visual Basic .NET:

```
Dim cmd As New AseCommand( _
   "titleid_proc", conn )
cmd.CommandType = CommandType.StoredProcedure
```

4 Add an AseParameter object to define the parameters for the stored procedure. You must create a new AseParameter object for each parameter the stored procedure requires:

For C#:

```
AseParameter param = cmd.CreateParameter();
param.ParameterName = "@title_id";
param.AseDbType = AseDbType.VarChar;
param.Direction = ParameterDirection.Input;
param.Value = "BU";
cmd.Parameters.Add( param );
```

For Visual Basic .NET:

```
Dim param As AseParameter = cmd.CreateParameter()
param.ParameterName = "@title_id"
param.AseDbType = AseDbType.VarChar
param.Direction = ParameterDirection.Input
param.Value = "BU"
cmd.Parameters.Add( param )
```

For more information about the Parameter object, see "AseParameter class" on page 156.

5 Call the ExecuteReader method to return the DataReader object. The Get methods are used to return the results in the desired datatype:

For C#:

```
AseDataReader reader = cmd.ExecuteReader();
while (reader.Read())
{
   string title = reader.GetString(0);
   string id = reader.GetString(1);
   decimal price = reader.GetDecimal(2);
   // do something with the data....
}
```

For Visual Basic .NET:

```
Dim reader As AseDataReader = cmd.ExecuteReader()
While reader.Read()
   Dim title As String = reader.GetString(0)
   Dim id As String = reader.GetString(1)
   Dim price As Decimal = reader.GetDecimal(2)
   ' do something with the data....
End While
```

6 Close the AseDataReader and AseConnection objects:

For C#:

```
reader.Close();
conn.Close();
```

For Visual Basic .NET:

```
reader.Close()
conn.Close()
```

Alternate way to call a stored procedure

You can also call a stored procedure using call syntax. This syntax is compatible with ODBC and JDBC. For example:

```
AseCommand cmd = new AseCommand("{ call
sp_product_info(?) }", conn);
```

In this case, do not set the Command type to CommandType.StoredProcedure. This syntax is available when you do not use named parameters and have set the AseCommand.NamedParameters property to "false."

For information about calling stored procedures that return a result set or a single value, see "Getting data using the AseCommand object" on page 37.

For information about calling stored procedures that do not return a result set, see “Inserting, updating, and deleting rows using the AseCommand object” on page 42.

Transaction processing

With Adaptive Server ADO.NET Data Provider, you can use the AseTransaction object to group statements together. Each transaction ends with a COMMIT or ROLLBACK, which either makes your changes to the database permanent or cancels all the operations in the transaction, respectively. When the transaction is complete, you must create a new AseTransaction object to make further changes. This behavior is different from ODBC and Embedded SQL, where a transaction persists after you execute a COMMIT or ROLLBACK until the transaction is closed.

If you do not create a transaction, Adaptive Server ADO.NET Data Provider operates in autocommit mode by default. There is an implicit Commit after each insert, update, or delete, and when an operation is completed, the change is made to the database. In this case, the changes cannot be rolled back.

For more information about the AseTransaction object, see “AseTransaction class” on page 170.

Setting the isolation level for transactions

You can choose to specify an isolation level when you begin the transaction. The isolation level applies to all commands executed within the transaction.

For more information about isolation levels, see the *Adaptive Server Enterprise Performance and Tuning Guide*.

The locks that Adaptive Server uses when you enter a Select statement depend on the transaction's isolation level.

The following example uses an AseTransaction object to issue and then roll back a SQL statement. The transaction uses Isolation level 2 (RepeatableRead), which places a Write lock on the row being modified so that no other database user can update the row.

❖ **Using an AseTransaction object to issue a command**

1 Declare and initialize an AseConnection object:

For C#:

```
AseConnection conn = new AseConnection(
   c_connStr );
```

For Visual Basic .NET:

```
Dim conn As New AseConnection( _
   c_connStr )
```

2 Open the connection:

For C#:

```
conn.Open();
```

For Visual Basic .NET:

```
conn.Open()
```

3 Issue a SQL statement to change the price of “Tee shirts”:

For C#:

```
string stmt = "update product " +
   " set unit_price = 2000.00 " +
   " where name = 'Tee shirt'";
```

For Visual Basic .NET:

```
Dim stmt As String = "update product " + _
   " set unit_price = 2000.00 " + _
   " where name = 'Tee shirt'"
```

4 Create an AseTransaction object to issue the SQL statement using a Command object.

Using a transaction allows you to specify the isolation level. Isolation level 2 (RepeatableRead) is used in this example so that another database user cannot update the row:

For C#:

```
AseTransaction trans = conn.BeginTransaction(
   IsolationLevel.RepeatableRead );
AseCommand cmd = new AseCommand( stmt, conn, trans );
int rows = cmd.ExecuteNonQuery();
```

For Visual Basic .NET:

```
Dim trans As AseTransaction = _
   conn.BeginTransaction( _
   IsolationLevel.RepeatableRead )
Dim cmd As New AseCommand( _
   stmt, conn, trans )
Dim rows As Integer = cmd.ExecuteNonQuery()
```

5 Roll back the changes:

For C#:

```
trans.Rollback();
```

For Visual Basic .NET:

```
trans.Rollback()
```

The AseTransaction object allows you to commit or roll back your changes to the database. If you do not use a transaction, Adaptive Server ADO.NET Data Provider operates in autocommit mode and you cannot roll back any changes that you make to the database. If you want to make the changes permanent, you would use the following:

For C#:

```
trans.Commit();
```

For Visual Basic .NET:

```
trans.Commit()
```

6 Close the AseConnection object:

For C#:

```
conn.Close();
```

For Visual Basic .NET:

```
conn.Close()
```

Error handling

Your application must be designed to handle any errors that occur, including ADO.NET errors. Handle ADO.NET errors within your code the same way that you handle other errors in your application.

When errors occur during execution, Adaptive Server ADO.NET Data Provider throws AseException objects. Each AseException object consists of a list of AseError objects, and these error objects include the error message and code. In addition, other exceptions are possible, such as IndexOutOfRangeException and NotSupportedException.

Errors are different from conflicts, which occur when changes are applied to the database. Your application should include a process to compute correct values or to log conflicts when they arise.

Adaptive Server ADO.NET Data Provider error-handling example

The following example is from the Simple sample project. Any errors that occur during execution and that originate with Adaptive Server ADO.NET Data Provider objects are handled by displaying them in a message box. The following code catches the error and displays its message.

For C#:

```
catch( AseException ex )
{
   MessageBox.Show( ex.Message );
}
```

For Visual Basic .NET:

```
Catch ex As AseException
   MessageBox.Show(ex.Message)
End Try
```

Connection error-handling example

The following example is from the Table Viewer sample project. If an error occurs when the application attempts to connect to the database, the following code uses a try-and-catch block to catch the error and display its message:

For C#:

```
try
{
   _conn = new AseConnection(
      txtConnectString.Text );
   _conn.Open();
}
catch( AseException ex )
{
   MessageBox.Show(ex.Message, "Failed to connect");
}
```

For Visual Basic .NET:

```
Try
   Dim  _conn As New AseConnection( _
      txtConnectString.Text )
   conn.Open()
Catch ex As AseException
   MessageBox.Show(ex.Message, "Failed to connect")
End Try
```

For more error-handling examples, see “Understanding the Simple sample project” on page 13 and “Understanding the Table Viewer sample project” on page 18.

For more information about error handling, see “AseException class” on page 153 and “AseError class” on page 150.

Performance consideration

This section provides tips on developing and deploying applications with the Adaptive ServerADO.NET Data Provider.

DbType.String vs. DbType.AnsiString

Although both DbType.String and DbType.AnsiString deal with character data, these datatypes are processed differently, and using the wrong data type can have a negative effect on the application’s performance. DbType.String identifies the parameter as a 2-byte Unicode value and is sent to the server as such. DbType.AnsiString causes the parameter to be sent as a multibyte character string. To avoid excessive string conversions, use:

- DbType.AnsiString for char or varchar columns and parameters.
- DbType.String for unichar and univarchar columns and parameters.

CHAPTER 4

Adaptive Server Advanced Features

This chapter describes the advanced Adaptive Server features you can use with ADO.NET Data Provider.

Supported Adaptive Server Cluster Edition features

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

For more information about the 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 automatically enabled 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

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

Connection failover allows a client application to switch to an alternate Adaptive Server if the primary server becomes unavailable due to an unplanned event, like power outage or a socket failure. In a cluster environment, client applications can 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

To enable Cluster Edition connection failover, set the HASession connection string property to 1. For example:

```
Data Source=server1;Port=port1;User ID=sa;Password=;
Initial Catalog=sdc;HASession=1;
AlternateServers=server2:port2,...,serverN:portN;
```

In this example, Data Source defines the primary server and port. ADO.NET Provider by Sybase attempts to establish connection to the primary server first and, if unsuccessful, goes through the servers listed in Alternate Servers until a connection is established or until the end of the list is reached.

Note The list of alternate servers specified in the connection string is used only during initial connection. After the connection is established with any available instance, and if 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.

Using Distributed Transactions

You can use the Adaptive Server ADO.NET Data Provider to participate in two-phase commit transactions. This feature requires the use of .NET Enterprise Services, which manages the distributed transactions.

Programming using Enterprise Services

Services in unmanaged code are known as COM+ services. The COM+ services infrastructure can be accessed from managed and unmanaged code. In .NET, these services are referred to as Enterprise Services. Working with transactions in Enterprise Services using ADO.NET is straightforward.

❖ **Programming using Enterprise Services**

1. Derive the components from *System.EntrepriseService.ServicedComponent*.
2. Specify the custom attributes (such as Transaction, AutoComplete, and others) to specify the requested services and their options. For a complete list of the attributes, refer to the Enterprise Services documentation.

   Note The Timeout Option in the .NET Transaction attribute has to be explicitly set to *-1* or a very high number. .NET documentation states that the ADO.NET transaction timeout default is *0*, which means it will never time out. However, this actually causes an immediate transaction timeout, which rolls back the entire transaction.

3. Sign and build the assembly.
4. Register the assembly.

Connection properties for Distributed Transaction support

The following are the connection properties used in conjunction with Distributed Transaction support.

- Distributed Transaction Protocol (DistributedTransactionProtocol) – To specify the protocol used to support the distributed transaction, XA Interface standard, or MS DTC OLE Native protocol, set up the property DistributedTransactionProtocol=*OLE* native protocol in the connection string. The default protocol is *XA*.
- Tightly Coupled Transaction (TightlyCoupledTransaction) – When you have a distributed transaction using two resource managers that point to the same Adaptive Server server, you have a situation called a “Tightly Coupled Transaction.” Under these conditions, if you do not set this property to *1*, the Distributed Transaction may fail.

  To summarize, if you open two database connections to the same Adaptive Server server and enlist these connections in the same distributed transaction, you must set TightlyCoupledTransaction=*1*.

- Enlist – The AseConnection object automatically enlists in an existing distributed transaction if it determines that a transaction is active. Automatic transaction enlistment occurs when the connection is opened or retrieved from the connection pool. You can disable this auto-enlistment by specifying Enlist=0 as a connection string parameter for an AseConnection.

  If auto-enlistment is disabled, you can enlist in an existing distributed transaction by calling the EnlistDistributedTransaction method on the AseConnection with a passed-in ITransaction parameter that is a reference to an existing transaction. After calling the EnlistDistributedTransaction, all updates made using this instance of AseConnection will be made as part of this global transaction. As a result, it will be committed or rolled back when the global transaction is committed or rolled back.

  Note The AseConnection object must be open before calling EnlistDistributedTransaction.

  You can use EnlistDistributedTransaction when you pool business objects. If a business object is pooled with an open connection, automatic transaction enlistment occurs only when that connection is opened or pulled from the connection pool. If multiple transactions are performed using the pooled business object, the open connection for that object will *not* automatically enlist in newly initiated transactions. In this instance, you can disable automatic transaction enlistment for the AseConnection and then enlist the AseConnection in transactions using EnlistDistributedTransaction

  Warning! EnlistDistributedTransaction returns an exception if the AseConnection has already begun a transaction either by using BeginTransaction or by executing the BEGIN TRANSACTION statement explicitly with an AseCommand.

Directory services

With directory services, Adaptive Server ADO.NET Data Provider can get connection and other information from a central LDAP server, to connect to an Adaptive Server server. It uses Directory Service URL (DSURL), which indicates which LDAP server to use.

LDAP as a directory service

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

The LDAP server can be located on a different platform from the one on which Adaptive Server or the clients are running. LDAP defines the communication protocol and the contents of messages exchanged between clients and servers. The LDAP server can store and retrieve information about:

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

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

The LDAP server can be configured with these access restrictions:

- Anonymous authentication – All data is visible to any user.
- User name and password authentication – Data Provider uses the user name and password supplied in the DSURL or in the *DSPrincipal* and *DSPassword* properties of the ConnectString.

Using directory services

To use directory services, add the following properties to the ConnectString:

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

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

To support high availability on the LDAP server, the DSURL accepts multiple URLs. Separate each URL with a semicolon. For example:

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

An example of DSURL follows:

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

where:

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

Password encryption

By default, the Adaptive Server ADO.NET Data Provider sends plain text passwords over the network to Adaptive Server for authentication. However, the Adaptive Server ADO.NET Data Provider also supports symmetrical and asymmetrical password encryption, and you can use this feature to 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 a different 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.

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

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 encryption, you require a server that supports this type of encryption, such as Adaptive Server 15.0.2.

Example

In this example, `sapass` is not sent over the wire until a login is negotiated and the password is encrypted and sent.

```
AseConnection.ConnectionString=
”Data Source=MANGO;" +
   "Port = 5000;" +
   "Database=pubs2;" +
   "UID=sa;" +
   "PWD=sapass;" +
   "EncryptPassword=1;”;
```

Using SSL

Secure Sockets Layer (SSL) is an industry standard for sending wire- or socket-level encrypted data over client-to-server and server-to-server connections.

SSL handshake

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

1. The client sends a connection request to the server. The request includes the SSL (or Transport Layer Security, TLS) options that the client supports.
2. The server returns its certificate and a list of supported CipherSuites, which includes SSL/TLS support options, the algorithms used for key exchange, and digital signatures.
3. A secure, encrypted session is established when both client and server have agreed upon a CipherSuite.

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.

Performance

Additional overhead 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 may make user login 10 to 20 times slower.

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

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

The Adaptive Server ADO.NET Data Provider and Adaptive Server support the CipherSuites that are available with the SSL Plus library API and the cryptographic engine, Security Builder, both from Certicom Corp.

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

From strongest to weakest, the supported CipherSuites in Adaptive Server ADO.NET Data Provider include:

- 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

SSL in Adaptive Server ADO.NET Data Provider

SSL provides the following levels of security:

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

Validating the server by its certificate

Any Adaptive Server ADO.NET Data 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 ADO.NET Data Provider client applications establish a socket connection to Adaptive Server the same 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.

For more information, see the *Open Client Client Library/C Reference Manual*.

The trusted roots file

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

The application program specifies the location of the trusted roots file using the TrustedFile=*trusted file path* property in the ConnectString. A trusted roots file with most widely-used CAs (Thawte, Entrust, Baltimore, VeriSign and RSA) is located in *%SYBASE%\ini\trusted.txt*.

Enabling SSL connections

To enable SSL for the Data Provider, add `Encryption=ssl; TrustedFile=<trusted file>` to the ConnectionString property. AseConnection then negotiates an SSL connection with the Adaptive Server server, for example:

```
AseConnection.ConnectionString=
     "Data Source=MANGO;" +
     "Port = 5000;" +
     "Database=pubs2;" +
     "UID=sa;" +
     "PWD=sapass;" +
     "Encryption=ssl;" +
     "TrustedFile='c:\sybase\ini\trusted.txt';";
```

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

Using failover in a high-availability system

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

Failover enables Adaptive Server to work in a high availability cluster in 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. Failover can be enabled by setting the connection property HASession to “1” (default value is “0”). If this property is not set, the session failover does not occur even if the server is configured for failover. You must also set the SecondaryServer and SecondaryPort properties.

See the Adaptive Server document, *Using Sybase Failover in a High Availability System*, for information about configuring your system for high availability.

If failover happens within a transaction, only changes that were committed to the database before failover are retained. When a failover occurs, the Provider tries to reconnect to the secondary server. If a connection to the secondary server is established, ADO.NET Data Provider throws an AseFailoverException with a message that failover has occurred. Then, the client must reapply the failed transaction with the new connection. If the connection to the secondary server is not established, a regular AseException is raised in ADO.NET Data Provider with a message that the connection has been lost. For example:

```
AseConnection.ConnectionString =
   ”Data Source='tpsun1';" +
   "Port = 5000;" +
   "Database=pubs2;" +
   "User ID=sa;" +
   "Password=sapass;" +
   "HASession=1;" +
   "Secondary Data Source='tpsun2';" +
   "Secondary Server Port=5000";
```

The following code shows how to catch AseFailoverException:

```
....
Open connection
...more code

try
{
   using (AseDataReader rdr =
   selectCmd.ExecuteReader())
   {
   ....
   }
}
catch (AseFailoverException)
{
   //Make sure that you catch AseFailoverException
   //before AseException as AseFailoverException is
   //derived from AseException

   //HA has occured. The application has successfully
   //connected to the secondary server. All uncommitted
   //transactions have been rolled back.

   //You could retry your transactions or prompt user
   //for a retry operation
}
```

```
catch (AseException)
{
//Either some other problem or the Failover did not
//successfully connect to the secondary server. Apps.
//should react accordingly
}
```

Using Kerberos authentication

Kerberos is an industry standard network authentication system that provides simple login authentication as well as mutual login 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 ADO.NET Data Provider provide support for Kerberos connections. The Adaptive Server ADO.NET Data Provider specifically supports MIT, CyberSafe, and Active Directory Key Distribution Centers (KDCs).

Process overview

The Kerberos authentication process works as follows:

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

5 The client sends the authentication packet and the service packet to the service.

6 The service decrypts the service packet to get the session key and decrypts the authentication packet to get the user information.

7 The service compares the user information from the authentication packet with the user information that was also contained in the service packet. If the two match, the user has been authenticated.

8 The service creates a confirmation packet that contains service specific information as well as validation data contained in the authentication packet.

9 The service encrypts this data with the session key and returns it to the client.

10 The client uses the session key obtained from the user packet it received from Kerberos to decrypt the packet and validates that the service is what it claims to be.

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

Requirements

To use Kerberos as an authentication system, you must configure Adaptive Server Enterprise to delegate authentication to Kerberos. See the *Adaptive Server Enterprise System Administration Guide* for more information. On Windows, the Kerberos client library comes installed with the client library.

To use Kerberos with the Adaptive Server ADO.NET Data Provider, you must have the MIT/CyberSafe Client library configured and enable Adaptive Server for Kerberos.

Enabling Kerberos authentication

To enable Kerberos for the Adaptive Server ADO.NET Data Provider, 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 or the principal as configured in the Key Distribution Center (KDC).

The Adaptive Server ADO.NET Data Provider will use this information to negotiate a Kerberos authentication with the configured KDC and Adaptive Server. On Windows, you might want to choose activedirectory to avoid any additional setup.

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

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

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

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

Enabling Kerberos on Windows

Add the following properties to your *AseConnection.ConnectionString:*

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

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

Obtaining an initial ticket from the Key Distribution Center

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

❖ **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. Verify that you have a TGT by entering at the command line:

```
% klist
```

   The results of the klist command should be:

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

Explanation of results

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

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

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

CHAPTER 5

Adaptive Server ADO.NET Data Provider API Reference

This chapter describes the APIs for Adaptive Server ADO.NET Data Provider. For the most recent API documentation and for information about properties and methods that are implementation of the Microsoft ADO.NET interfaces, see the Adaptive Server ADO.NET online help. To access the online help, open the Microsoft Document Explorer and navigate to “Sybase ASE ADO.NET Data Provider”. You can also find more information and examples in the Microsoft .NET Framework documentation.

AseClientPermission class

Description — Enables the Adaptive Server ADO.NET Data Provider to ensure that a user has a security level adequate to access an Adaptive Server Enterprise data source.

Base classes — DBDataPermission

AseClientPermission constructors

Description — Initializes a new instance of the AseClientPermission class.

Syntax 1 — void **AseClientPermission( )**

Syntax 2 — void **AseClientPermission(** PermissionState *state* **)**

Syntax 3 — void **AseClientPermission(** PermissionState *state*, bool *allowBlankPassword* **)**

Parameters — **state** One of the PermissionState values.

allowBlankPassword Indicates whether a blank password is allowed.

AseClientPermissionAttribute class

Description — Associates a security action with a custom security attribute.

Base classes — DBDataPermissionAttribute

AseClientPermissionAttribute constructor

Description — Initializes a new instance of the AseClientPermissionAttribute class.

Syntax — void **AseClientPermissionAttribute(** SecurityAction *action* **)**

Parameters **action** One of the SecurityAction values representing an action that can be performed using declarative security.

Return Value An AsePermissionAttribute object.

CreatePermission method

Description Returns an AsePermission object that is configured according to the attribute properties.

Syntax IPermission **CreatePermission( )**

AseCommand class

Description Represents commands performed on the Adaptive Server and encapsulates either dynamic SQL statements or stored procedures. It provides the following methods to execute commands on the Adaptive Server Database:

- ExecuteNonQuery – Execute command that does not return a resultset
- ExecuteScalar – Execute command that does not return a resultset
- ExecuteReader - Execute command that returns a single value
- ExecuteXmlReader - Execute command that returns XML

Base classes Component

Implements IDbCommand, IDisposable

See also “Using AseCommand to retrieve and manipulate data” on page 37 and “Accessing and manipulating data” on page 36.

Note If you are calling a stored procedure that takes parameters, you must specify the parameters for the stored procedure. For more information, see “Using stored procedures” on page 73 and “AseParameter class” on page 156.

AseCommand constructors

Description Initializes an AseCommand object.

Syntax 1	void **AseCommand( )**
Syntax 2	void **AseCommand(** string *cmdText* **)**
Syntax 3	void **AseCommand(** string *cmdText*, AseConnection *connection* **)**
Syntax 4	void **AseCommand(** string *cmdText*, AseConnection *connection*, AseTransaction *transaction* **)**
Parameters	**cmdText:** The text of the SQL statement or stored procedure.
	connection: The current connection.
	transaction: The AseTransaction in which the AseConnection executes.

Cancel method

Description	Cancels the execution of an AseCommand object.
Syntax	void **Cancel( )**
Usage	If there is nothing to cancel, nothing happens. If there is a command in process and the attempt to cancel fails, no exception is generated.
Implements	IDbCommand.Cancel

CommandText property

Description	The text of a SQL statement or stored procedure.
Syntax	string **CommandText**
Access	Read-write
Property value	The SQL statement or the name of the stored procedure to execute. The default is an empty string.
Implements	IDbCommand.CommandText
See also	“AseCommand constructors” on page 101.

CommandTimeout property

Description	The wait time in seconds before terminating an attempt to execute a command and generating an error.
Syntax	int **CommandTimeout**

Access	Read-write
Implements	IDbCommand.CommandTimeout
Default	30 seconds
Usage	A value of 0 indicates no limit. This should be avoided because it can cause the attempt to execute a command to wait indefinitely.

CommandType property

Description	The type of command represented by an AseCommand.
Syntax	CommandType **CommandType**
Access	Read-write
Implements	IDbCommand.CommandType
Usage	Supported command types are as follows: • **CommandType.StoredProcedure:** When you specify this CommandType, the command text must be the name of a stored procedure and you must supply any arguments as AseParameter objects. • **CommandType.Text:** This is the default value. When the CommandType property is set to StoredProcedure, the CommandText property should be set to the name of the stored procedure. The command executes this stored procedure when you call one of the Execute methods.

Connection property

Description	The connection object to which the AseCommand object applies.
Syntax	AseConnection **Connection**
Access	Read-write
Default	The default value is a null reference. In Visual Basic it is “Nothing.”

CreateParameter method

Description	Provides an AseParameter object for supplying parameters to AseCommand objects.

Syntax

AseParameter **CreateParameter( )**

Return value

A new parameter, as an AseParameter object.

Usage

- Stored procedures and some other SQL statements can take parameters, indicated in the text of a statement by the *@name* parameter.
- The CreateParameter method provides an AseParameter object. You can set properties on the AseParameter to specify the value, datatype, and so on for the parameter.

See also

"AseParameter class" on page 156.

ExecuteNonQuery method

Description

Executes a statement that does not return a result set, such as an Insert, Update, Delete, or a data definition statement.

Syntax

int **ExecuteNonQuery( )**

Return value

The number of rows affected.

Implements

IDbCommand.ExecuteNonQuery

Usage

- You can use ExecuteNonQuery to change the data in a database without using a DataSet. Do this by executing Update, Insert, or Delete statements.
- Although ExecuteNonQuery does not return any rows, output parameters or return values that are mapped to parameters are populated with data.
- For Update, Insert, and Delete statements, the return value is the number of rows affected by the command. For all other types of statements, the return value is -1.

ExecuteReader method

Description

Executes a SQL statement that returns a result set.

Syntax 1

AseDataReader **ExecuteReader( )**

Syntax 2

AseDataReader **ExecuteReader(CommandBehavior** *behavior* **)**

Parameters

behavior: One of CloseConnection, Default, KeyInfo, SchemaOnly, SequentialAccess, SingleResult, or SingleRow.

The default value is CommandBehavior.Default. For more information about this parameter, see the .NET Framework documentation for CommandBehavior Enumeration.

Return value	The result set as an AseDataReader object.
Usage	The statement is the current AseCommand object, with CommandText and Parameters as needed. The AseDataReader object is a read-only, forward-only result set. For modifiable result sets, use an AseDataAdapter class.
See also	"AseDataAdapter class" on page 126 and "AseDataReader class" on page 134.

ExecuteScalar method

Description	Executes a statement that returns a single value. If this method is called on a query that returns multiple rows and columns, only the first column of the first row is returned.
Syntax	object **ExecuteScalar( )**
Return Value	The first column of the first row in the result set, or a null reference if the result set is empty.
Implements	IDbCommand.ExecuteScalar

ExecuteXmlReader method

Description	Executes a SQL statement with a valid FOR XML clause, that returns a result set. ExecuteXmlReader can also be called with a statement that returns one text column that contains XML.
Syntax 1	XmlReader **ExecuteXmlReader( )**
Parameters	None.
Return value	The result set as an XmlReader object.
Usage	The statement is the current AseCommand object, with CommandText and Parameters as needed.
See also	XmlReader in the Microsoft .NET documentation.

NamedParameters

Description	This property governs the default behavior of the AseCommand objects associated with this connection.
Syntax	bool **NamedParameters**

Property value

The default value is the same as what is set on the associated AseConnection object. If this property is turned to “true” (the default on AseConnection), the Provider assumes that the user is not using parameter markers ('?') and instead using parameters by name. For example:

```
select total_sales from titles where title_id =
@title_id
```

Set this property to false if you want to use parameter markers. This is compatible with ODBC and JDBC.

For example:

```
select total_sales from titles where title_id = ?
```

Access

Read-write

Parameters property

Description

A collection of parameters for the current statement. Use the *@name* parameter or question marks in the CommandText to indicate parameters.

Syntax

AseParameterCollection **Parameters**

Access

Read-only

Property Value

The parameters of the SQL statement or stored procedure. The default value is an empty collection.

Usage

- When CommandType is set to Text, use the *@name* parameter. For example:

  ```
  SELECT * FROM Customers WHERE CustomerID = @name
  ```

  Or, if NamedParameters is set to “false” use ? parameter markers. For example:

  ```
  SELECT * FROM Customers WHERE CustomerID = ?
  ```

- If you use the question mark placeholder, the order in which AseParameter objects are added to the AseParameterCollection must directly correspond to the position of the question mark placeholder for the parameter in the command text.
- When the parameters in the collection do not match the requirements of the query to be executed, an error can result or an exception can be thrown.
- You have to specify the AseDbType for output parameters (whether prepared or not).

See also "AseParameterCollection class" on page 161.

Prepare method

Description Prepares or compiles the AseCommand on the data source.

Syntax void **Prepare( )**

Implements IDbCommand.Prepare

Usage

- Before you call Prepare, specify the datatype of each parameter in the statement to be prepared.
- If you call an Execute method after calling Prepare, any parameter value that is larger than the value specified by the Size property is automatically truncated to the original specified size of the parameter, and no truncation errors are returned.

Transaction property

Description Associates the current command with a transaction.

Syntax AseTransaction **Transaction**

Access Read-write

Usage The default value is a null reference. In Visual Basic this is Nothing.

You cannot set the Transaction property if it is already set to a specific value and the command is executing. If you set the transaction property to an AseTransaction object that is not connected to the same AseConnection as the AseCommand object, an exception will be thrown the next time you attempt to execute a statement.

UpdatedRowSource property

Description Command results that are applied to the DataRow when used by the Update method of the AseDataAdapter.

Syntax UpdateRowSource **UpdatedRowSource**

Access Read-write

Implements IDbCommand.UpdatedRowSource

Property value	One of the UpdatedRowSource values. If the command is automatically generated, the default is None. Otherwise, the default is “Both.”

AseCommandBuilder class

Description	Automatically builds SQL INSERT, UPDATE, and DELETE statements for a single-table based on a SQL SELECT statement.
Base classes	Component
Implements	IDisposable
See also	“AseDataAdapter class” on page 126.

AseCommandBuilder constructors

Description	Initializes an AseCommandBuilder object.
Syntax 1	void **AseCommandBuilder( )**
Syntax 2	void **AseCommandBuilder(** AseDataAdapter *adapter* **)**
Parameters	**adapter** An AseDataAdapter object for which to generate reconciliation statements.

DataAdapter property

Description	The AseDataAdapter for which to generate statements.
Syntax	AseDataAdapter **DataAdapter**
Access	Read-write
Property value	An AseDataAdapter object.
Usage	When you create a new instance of AseCommandBuilder, any existing AseCommandBuilder that is associated with this AseDataAdapter is released.

DeleteCommand property

Description
An AseCommand object that is executed against the database when Update() is called to delete rows in the database that correspond to deleted rows in the DataSet.

Syntax
AseCommand **DeleteCommand**

Access
Read-write

Usage
When DeleteCommand is assigned to an existing AseCommand object, the AseCommand object is not cloned; the DeleteCommand maintains a reference to the existing AseCommand.

DeriveParameters method

Description
Populates the Parameters collection of the specified AseCommand object. This is used for the stored procedure specified in the AseCommand.

Syntax
void **DeriveParameters(** AseCommand *command* **)**

Parameters
command An AseCommand object for which to derive parameters.

Usage
- DeriveParameters overwrites any existing parameter information for the AseCommand.
- DeriveParameters requires an extra call to the database server. If the parameter information is known in advance, it is more efficient to populate the Parameters collection by setting the information explicitly.

Dispose method

Description
Frees the resources associated with the object.

Syntax
void **Dispose( )**

GetDeleteCommand method

Description
A generated AseCommand object, performs Delete operations on the database when AseDataAdapter.Update() is called.

Syntax
AseCommand **GetDeleteCommand( )**

Return value
The automatically generated AseCommand object required to perform deletions.

Usage

- The GetDeleteCommand method returns the AseCommand object to be executed, so it may be useful for informational or troubleshooting purposes.
- You can also use GetDeleteCommand as the basis of a modified command. For example, you might call GetDeleteCommand and modify the CommandTimeout value, and then explicitly set that value on the AseDataAdapter.
- SQL statements are first generated when the application calls Update or GetDeleteCommand. After the SQL statement is first generated, the application must explicitly call RefreshSchema if it changes the statement in any way. Otherwise, the GetDeleteCommand will still be using information from the previous statement.

GetInsertCommand method

Description

A generated AseCommand object, performs Insert operations on the database when an AseDataAdapter.Update() is called.

Syntax

AseCommand **GetInsertCommand( )**

Return value

The automatically generated AseCommand object required to perform insertions.

Usage

- The GetInsertCommand method returns the AseCommand object to be executed, so it may be useful for informational or troubleshooting purposes.
- You can also use GetInsertCommand as the basis of a modified command. For example, you can call GetInsertCommand and modify the CommandTimeout value, and then explicitly set that value on the AseDataAdapter.
- SQL statements are generated either when the application calls Update or GetInsertCommand. After the SQL statement is first generated, the application must explicitly call RefreshSchema if it changes the statement in any way. Otherwise, GetInsertCommand will be still be using information from the previous statement, which might not be correct.

GetUpdateCommand method

Description

A generated AseCommand object, performs Update operations on the database when an AseDataAdapter.Update() is called.

Syntax

AseCommand **GetUpdateCommand( )**

Return value

The automatically generated AseCommand object required to perform updates.

Usage

- The GetUpdateCommand method returns the AseCommand object to be executed, so it may be useful for informational or troubleshooting purposes.
- You can also use GetUpdateCommand as the basis of a modified command. For example, you might call GetUpdateCommand and modify the CommandTimeout value, and then explicitly set that value on the AseDataAdapter.
- SQL statements are first generated when the application calls Update or GetUpdateCommand. After the SQL statement is first generated, the application must explicitly call RefreshSchema if it changes the statement in any way. Otherwise, the GetUpdateCommand will be still be using information from the previous statement, which might not be correct.

InsertCommand property

Description

An AseCommand that is executed against the database when an Update() is called that adds rows to the database to correspond to rows that were inserted in the DataSet.

Syntax

AseCommand **InsertCommand**

Access

Read-write

Usage

When InsertCommand is assigned to an existing AseCommand object, the AseCommand is not cloned. The InsertCommand maintains a reference to the existing AseCommand.

If this command returns rows, the rows can be added to the DataSet depending on how you set the UpdatedRowSource property of the AseCommand object.

See also

“Update method” on page 133

PessimisticUpdate property

Description

Indicates whether to implement pessimistic or optimistic update.

Syntax
public bool PessimisticUpdate

Access
Read-write

Property Value
True for pessimistic update. False for optimistic update.

Usage
- Pessimistic update locks a record such that the record is viewable by all users but is editable only to one user
- Optimistic update allows multiple users to edit the same record.

QuotePrefix property

Description
The beginning character or characters to use when specifying database object names that contain characters such as spaces.

Syntax
string **QuotePrefix**

Access
Read-write

Property value
The beginning character or characters to use. This can be square brackets, or, if the ASE QUOTED_IDENTIFIER option is set to “off”, it can be double quotes. The default is an empty string.

Usage
- ASE objects can contain characters such as spaces, commas, and semicolons. The QuotePrefix and QuoteSuffix properties specify delimiters to encapsulate the object name.
- Although you cannot change the QuotePrefix or QuoteSuffix properties after an Insert, Update, or Delete operation, you can change their settings after calling the Update method of a DataAdapter.

See also
- The Adaptive Server Enterprise *Reference Manual* for more information about identifiers.
- The Adaptive Server Enterprise *Reference Manual* for more information about the QUOTED IDENTIFIER option.

QuoteSuffix property

Description
The ending character or characters to use when specifying database objects whose names contain characters such as spaces.

Syntax
string **QuoteSuffix**

Access
Read-write

Property value

The ending character or characters to use. This can be square brackets, or, if the ASE QUOTED_IDENTIFIER option is set to off, it can be double quotes. The default is an empty string.

Usage

- ASE objects can contain characters such as spaces, commas, and semicolons. The QuotePrefix and QuoteSuffix properties specify delimiters to encapsulate the object name.
- Although you cannot change the QuotePrefix or QuoteSuffix properties after an Insert, Update, or Delete operation, you can change their settings after calling the Update method of a DataAdapter.

See also

- The Adaptive Server Enterprise *Reference Manual* for more information about identifiers.
- The Adaptive Server Enterprise *Reference Manual* for more information about the QUOTED IDENTIFIER option.

RefreshSchema method

Description

Refreshes the database schema information used to generate Insert, Update, or Delete statements.

Syntax

void **RefreshSchema( )**

Usage

Call RefreshSchema whenever the SelectCommand value of the associated AseDataAdapter changes.

SelectCommand property

Description

An AseCommand that is used during Fill or FillSchema to obtain a result set from the database for copying into a DataSet.

Syntax

AseCommand **SelectCommand**

Access

Read-write

Usage

- When SelectCommand is assigned to a previously created AseCommand, the AseCommand is not cloned. The SelectCommand maintains a reference to the previously created AseCommand object.
- If the SelectCommand does not return any rows, no tables are added to the DataSet, and no exception is raised.
- The Select statement can also be specified in the AseDataAdapter constructor.

UpdateCommand property

Description	An AseCommand that is executed against the database when Update() is called to update rows in the database that correspond to updated rows in the DataSet.
Syntax	AseCommand **UpdateCommand**
Access	Read-write
Usage	• When UpdateCommand is assigned to a previously created AseCommand, the AseCommand is not cloned. The UpdateCommand maintains a reference to the previously created AseCommand object. • If execution of this command returns rows, these rows can be merged with the DataSet depending on how you set the UpdatedRowSource property of the AseCommand object.
See also	"Update method" on page 133.

AseConnection class

Description	Represents a connection to an Adaptive Server database.
Base classes	Component
Implements	IDbConnection, IDisposable
See also	"Connecting to a database" on page 31.

AseConnection constructors

Description	Initializes an AseConnection object. The connection must then be opened before you can carry out any operations against the database.
Syntax 1	**AseConnection( )**
Syntax 2	**AseConnection(** string *connectionString* **)**
Parameters	**connectionString:** An Adaptive Server connection string. A connection string is a semicolon-separated list of keyword-value pairs.

Table 5-1: Connection string parameters

Property name	Description	Required	Default value
UID, UserID, User ID, User	A case-sensitive user ID required to connect to the Adaptive Server server.	Yes	Empty
PWD, Password	A case-sensitive password to connect to the Adaptive Server server.	No, if the user name does not require a password	Empty
Server, Data Source, DataSource, Address, Addr, Network Address, Server Name	The name or the IP address of the Adaptive Server server.	Yes	Empty
Port, Server Port	The port number of Adaptive Server server.	Yes, unless the port number is specified in the Datasource	Empty
AnsiNull	Strict ODBC compliance where you cannot use “= NULL”. Instead you have to use “IsNull”. Set to 1 if you want to change the default behavior.	No	0
ApplicationName, Application Name	The name to be used by Adaptive Server to identify the client application	No	Empty.
BufferCacheSize	Keeps the Input / Output buffers in pool. Increase for very large results to boost performance	No	20
CharSet	The designated character set. The Adaptive Server ADO.NET Data Provider by default negotiates the same default character set as Adaptive Server. The default character set is ServerDefault.	No	Value of ServerDef ault
ClientHostName	The name of client host passed in the login record to the server, for example: `ClientHostName='MANGO'`	No	Empty
ClientHostProc	The identity of client process on this host machine passed in the login record to the server, for example: `ClientHostProc='MANGO-PROC'`	No	Empty
CodePageType	Specifies the type of character encoding used. The valid values are ANSI and OEM.	No	ANSI

Property name	Description	Required	Default value
ConnectionIdleTimeout, Connection IdleTimeout, Connection Idle Timeout	The time, in seconds, that a connection can stay idle in the connection pool before the driver closes the connection. A value of 0 allows connections to stay idle for an indefinite amount of time.	No.	0
Connection Lifetime	The time, in seconds, that connections can stay open. When a client closes a connection that has reached or exceeded the defined Connection Lifetime, before the driver closes the connection instead of returning it to the connection pool. An idle connection is closed and removed from the connection pool once it reaches the defined Connection Lifetime. The default value of Connection Lifetime is 0, which indicates that the connection can remain open for an indefinite amount of time.	No	0
CumulativeRecordCount,Cumulative Record Count,CRC	By default the driver (use provider for ADO.NET) returns the total records updated when multiple update statements are executed in a stored procedure. This count includes 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
Database, Initial Catalog	The database to which you want to connect.	No	Empty.
DSPassword, Directory Service Password	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, Directory Service Principal	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.

Property name	Description	Required	Default value
DSURL, Directory Service URL	The URL to the LDAP server.	No	Empty.
DTCProtocol (Windows only)	Allows the driver to use either an XA protocol or OleNative protocol when using distributed transactions. See "Using Distributed Transactions" on page 83, in Chapter 4, "Adaptive Server Advanced Features."	No	XA
EnableServerPacketSize	Allows Adaptive Server server versions 15.0 or later to choose the optimal packetsize.	No	1
EncryptPassword, EncryptPwd, Encrypt Password	Specifies if password encryption is enabled: 0 indicates password encryption is disabled, 1 indicates password encryption is enabled.	No	0
Encryption	The designated encryption. Possible values: ssl, none.	No	Empty.
FetchArraySize	Specifies the number of rows the driver retrieves when fetching results from the server.	No	25
HASession	Specifies if High Availability is enabled. 0 indicates High Availability disabled, 1 High Availability enabled.	No	0
IgnoreErrorsIfRS Pending	Specifies whether the driver is to continue processing or stop if error messages are present. When set to 1, the driver will ignore errors & continue processing the results if more results are available from the server. When set to 0, the driver will stop processing the results if an error is encountered even if there are results pending.	No	0
Language	The language in which Adaptive Server returns error messages.	No	Empty – Adaptive Server uses English by default.

Property name	Description	Required	Default value
LoginTimeOut, Connect Timeout, Connection Timeout	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
min pool size	You can force the provider to close connections to Adaptive Server so that total number of open connections hover around min pool size. The provider closes the connection on AseConnection.Close() if the number of connections in the pool are equal to min pool size.	No	20
max pool size	You can restrict the connection pool not to grow more than the specified max pool size. The provider will throw an AseException on AseConnection.Open() if this limit is reached.	No	100
NamedParameters	Set to false if you intend to use parameter markers instead of named parameters. Example with named parameters the SQL will look like `SELECT *` from titles where `title_id = @title_id`. The same SQL with parameter markers will look like `SELECT *` from titles where `title_id = ?`.	No	true
PacketSize, Packet Size	The number of bytes per network packet transferred between Adaptive Server and the client.	No	512
Ping Server	Set to false if you do not want the provider to verify that the connection is valid before it uses it from the connection pool.	No	true
pooling	To disable connection pooling set to false.	No	true
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

Property name	Description	Required	Default value
RestrictMaximum PacketSize	If the you have memory constraints when EnableServerPacketSize is set to 1, then set this property to an int value in multiples of 512 to a maximum of 65536.	No	0
SecondaryPort, Secondary Port, Secondary Server Port	The port number of the Adaptive Server server acting as a failover server in an active-active or active-passive setup.	Yes, if HASession is set to 1, unless the port number is specified in the Secondary DataSource.	Empty.
SecondaryServer, Secondary Data Source, Secondary DataSource, Secondary Server, Secondary Address, Secondary Addr, Secondary Network Address	The name or the IP address of the Adaptive Server acting as a failover server in an active-active or active-passive setup. "Secondary Data Source" can be: SecondaryServer:SecondaryPort or SecondaryServer, SecondaryPort.	Yes, if HASession is set to 1.	Empty.
SQL Initialization String, SQLInitializationString, SQL Init String , SQLInitString, Initialization String, InitializationString	Defines a space-separated list of commands that is passed to database servers and executed immediately after connection. The commands must be SQL commands that are not used to query results.	No	Empty
TextSize	The maximum size of binary or text data in bytes that will be sent to or received from Adaptive Server, for example: `TextSize=64000` sets this limit to 64K bytes.	No	Empty. Adaptive Server default is 32K.
TightlyCoupled Transaction (Windows only)	When using distributed transactions, if you are using two DSNs which connect to the same Adaptive Server server, set this to 1. See "Using Distributed Transactions" on page 83, in Chapter 4, "Adaptive Server Advanced Features."	No	0
TrustedFile	If Encryption is set to ssl, this property should be set to the path to the Trusted File.	No	Empty

Property name	Description	Required	Default value
UseCursor, Use Cursor	Specifies whether cursors are to be used by the driver. 0 indicates do not use cursors and 1indicates use cursors.	No	0

Note Data Source, DataSource, Secondary Data Source, Secondary DataSource are special keywords. In addition to being the server name, they can also be formatted as

```
DataSource=servername,port
```

or

```
DataSource=servername:port
```

For example, `DataSource=gamg:4100` sets the server name to “gamg” and the port to “4100.” In this case, the Port keyword would not be needed in the connection string.

Example

The following statement initializes an AseConnection object for a connection to a database named “policies” running on an Adaptive Server database server named “HR-001.” The connection uses a user ID of “admin” with a password of “money.”

```
"Data Source='HR-001';
Port=5000; UID='admin';
PWD='money';
Database='policies';"
```

BeginTransaction method

Description

Returns a transaction object. Commands associated with a transaction object are executed as a single transaction. The transaction is terminated with a Commit() or Rollback().

Syntax 1

AseTransaction **BeginTransaction( )**

Syntax 2

AseTransaction **BeginTransaction(** IsolationLevel *isolationLevel* **)**

Parameters

isolationLevel: A member of the IsolationLevel enumeration. The default value is ReadCommitted.

Return value

An object representing the new transaction.

Usage

To associate a command with a transaction object, use the AseCommand.Transaction property.

Example

```
AseTransaction tx = conn.BeginTransaction(IsolationLevel.ReadUncommitted );
```

See also

"Commit method" on page 170, "Rollback method" on page 171, "Transaction processing" on page 76, and the *Adaptive Server Enterprise System Administration Guide* for information about inconsistencies.

ChangeDatabase method

Description

Changes the current database for an open AseConnection.

Syntax

void **ChangeDatabase(** string *database* **)**

Parameters

database: The name of the database to use instead of the current database.

Implements

IDbConnection.ChangeDatabase

Close method

Description

Closes a database connection.

Syntax

void **Close( )**

Implements

IDbConnection.Close

Usage

The Close method rolls back any pending transactions. It then releases the connection to the connection pool, or closes the connection if connection pooling is disabled. If Close is called while handling a StateChange event, no additional StateChange events are fired. An application can call Close more than one time.

ConnectionString property

Description

A database connection string.

Syntax

string **ConnectionString**

Access

Read-write

Implements

IDbConnection.ConnectionString

Usage

- The ConnectionString is designed to match the ODBC connection string format as closely as possible.
- You can set the ConnectionString property only when the connection is closed. Many of the connection string values have corresponding read-only properties. When the connection string is set, all of these properties are updated. However, if an error is detected, none of the properties are updated. AseConnection properties return only those settings contained in the ConnectionString.
- If you reset the ConnectionString on a closed connection, all connection string values and related properties are reset, including the password.
- When the property is set, a preliminary validation of the connection string is performed. When an application calls the Open method, the connection string is fully validated. A runtime exception is generated if the connection string contains invalid or unsupported properties.
- Values can be delimited by single or double quotes. Either single or double quotes can be used within a connection string by using the other delimiter. For example, `name="value's"` or `name= 'value"s'`, but not `name='value's'` or `name= ""value""`.

  Blank characters are ignored unless they are placed within a value or within quotes.

  Keyword-value pairs must be separated by a semicolon. If a semicolon is part of a value, it must also be delimited by quotes.

  Escape sequences are not supported, and the value type is irrelevant.

  Names are not case sensitive. If a property name occurs more than once in the connection string, the value associated with the last occurrence is used.
- Use caution when constructing a connection string based on user input, such as when retrieving a user ID and password from a dialog box, and appending it to the connection string. The application should not allow a user to embed extra connection string parameters in these values.

Example

The following statements set a connection string to connect to an Adaptive Server database called pubs2 running on the server mango and opens the connection.

```
AseConnection conn = new AseConnection( "Data Source=mango;
Port=5000;
UID=sa;
PWD='';
Database='pubs2';
" ):
conn.Open();
```

ConnectionTimeout property

Description	The number of seconds before a connection attempt times out with an error.
Syntax	int **ConnectionTimeout**
Access	Read-only
Default	15 seconds.
Implements	IDbConnection.ConnectionTimeout
Example	The following statement changes the ConnectionTimeout to 30 seconds.

```
conn.ConnectionTimeout = 30;
```

CreateCommand method

Description	Initializes an AseCommand object. You can use the properties of the AseCommand to control its behavior.
Syntax	AseCommand **CreateCommand( )**
Return value	An AseCommand object.
Usage	The Command object is associated with the AseConnection.

Database property

Description	The name of the current database to be used after a connection is opened.
Syntax	string **Database**
Access	Read-only
Implements	IDbConnection.Database
Usage	`if (conn.Database != "pubs2")`

```
conn.ChangeDatabase("pubs2");
```

InfoMessage event

Description
Occurs when Adaptive Server ADO.NET Data Provider sends a warning or an informational message.

Syntax
event AseInfoMessageEventHandler **InfoMessage**

Usage
The event handler receives an argument of type AseInfoMessageEventArgs containing data related to this event. The Errors and Message properties provide information specific to this event.

NamedParameters

Description
Governs the default behavior of the AseCommand objects associated with this connection. For more information see the AseCommand class (NamedParameter property).

Syntax
bool **NamedParameters**

Property value
This can be either set by the ConnectString (NamedParameters='true'/'false') or the user can set it directly through an instance of AseConnection.

Access
Read-write

Open method

Description
Opens a connection to a database, using the previously-specified connection string.

Syntax
void **Open( )**

Implements
IDbConnection.Open

Usage

- The AseConnection draws an open connection from the connection pool if one is available. Otherwise, it establishes a new connection to the data source.
- If the AseConnection goes out of scope, it is not closed. Therefore, you must explicitly close the connection by calling Close or Dispose.

State property

Description	The current state of the connection.
Syntax	ConnectionState **State**
Access	Read-only
Default	Closed.
Implements	IDbConnection.State
See also	"Checking the connection state" on page 34.

StateChange event

Description	Occurs when the state of the connection changes.
Syntax	event StateChangeEventHandler **StateChange**
Usage	The event handler receives an argument of StateChangeEventArgs with data related to this event. Two StateChangeEventArgs properties provide information specific to this event: CurrentState and OriginalState.

TraceEnter, TraceExit events

Description	Traces database activity within an application for debugging.
Syntax	public delegate void **TraceEnterEventHandler**(AseConnection connection, object source, string method, object[] parameters); public delegate void **TraceExitEventHandler**(AseConnection connection, object source, string method, object returnValue);
Usage	Use TraceEnter and TraceExit events to hook up your own tracing method. This event is unique to an instance of a connection. This allows different connections to be logged to different files. It can ignore the event, or you can program it for other tracing. In addition, by using a .NET event, you can set up more than one event handler for a single connection object. This enables you to log the event to both a window and a file at the same time.

AseDataAdapter class

Description

The link between the DataSet class and Adaptive Server. It uses two important methods:

- Fill() – a DataSet with data from the server
- Update() – apply the changes in a DataSet back to the server

When using AseDataAdapter Fill or Update methods, it can open the connection if it is not already open.

Base classes

Component

Implements

IDbDataAdapter, IDisposable

Usage

The DataSet provides a way to work with data offline. The AseDataAdapter provides methods to associate a DataSet with a set of SQL statements.

See also

"Using AseDataAdapter to access and manipulate data" on page 49 and "Accessing and manipulating data" on page 36.

AseDataAdapter constructors

Description

Initializes an AseDataAdapter object.

Syntax 1

AseDataAdapter( )

Syntax 2

AseDataAdapter(AseCommand *selectCommand* **)**

Syntax 3

AseDataAdapter(string *selectCommandText*, AseConnection *selectConnection* **)**

Syntax 4

AseDataAdapter(string *selectCommandText*, string *selectConnectionString* **)**

Parameters

selectCommand: An AseCommand object that is used during Fill to select records from the data source for placement in the DataSet.

selectCommandText: A Select statement or stored procedure to be used by the SelectCommand property of the AseDataAdapter.

selectConnection: An AseConnection object that defines a connection to a database.

selectConnectionString: A connection string for an Adaptive Server database.

Example

The following code initializes an AseDataAdapter object:

```
AseDataAdapter da = new AseDataAdapter( "SELECT emp_id,
```

```
emp_lname FROM employee," conn );
```

AcceptChangesDuringFill property

Description
A value that indicates whether AcceptChanges is called on a DataRow after it is added to the DataTable.

Syntax
bool **AcceptChangesDuringFill**

Access
Read-write

Usage
When this property is "true," DataAdapter calls the AcceptChanges function on the DataRow. If "false," AcceptChanges is not called, and the newly added rows are treated as inserted rows. The default is "true."

ContinueUpdateOnError property

Description
A value that specifies whether to generate an exception when an error is encountered during a row update.

Syntax
bool **ContinueUpdateOnError**

Access
Read-write

Usage
- The default is "false." Set this property to true to continue the update without generating an exception.
- If ContinueUpdateOnError is "true," no exception is thrown when an error occurs during the update of a row. The update of the row is skipped and the error information is placed in the RowError property of the row. The DataAdapter continues to update subsequent rows.
- If ContinueUpdateOnError is "false," an exception is thrown when an error occurs.

DeleteCommand property

Description
An AseCommand object that is executed against the database when Update() is called to delete rows in the database that correspond to deleted rows in the DataSet.

Syntax
AseCommand **DeleteCommand**

Access
Read-write

Usage

When DeleteCommand is assigned to an existing AseCommand object, the AseCommand object is not cloned; the DeleteCommand maintains a reference to the existing AseCommand.

Fill method

Description

Adds or refreshes rows in a DataSet or DataTable object with data from the database.

Syntax 1

int **Fill(** DataSet *dataSet* **)**

Syntax 2

int **Fill(** DataSet *dataSet*, string *srcTable* **)**

Syntax 3

int **Fill(** DataSet *dataSet*, int *startRecord*, int *maxRecords*, string *srcTable* **)**

Syntax 4

int **Fill(** DataTable *dataTable* **)**

Parameters

dataSet: A DataSet to fill with records and, optionally, schema.

srcTable: The name of the source table to use for table mapping.

startRecord: The zero-based record number to start with.

maxRecords: The maximum number of records to be read into the DataSet.

dataTable: A DataTable to fill with records and, optionally, schema.

Return Value

The number of rows successfully added or refreshed in the DataSet.

Usage

- Even if you use the StartRecord argument to limit the number of records that are copied to the DataSet, all records in the AseDataAdapter query are fetched from the database to the client. For large result sets, this can have a significant performance impact.
- An alternative is to use an AseDataReader when a read-only, forward-only result set is sufficient, perhaps with SQL statements (ExecuteNonQuery) to carry out modifications. Another alternative is to write a stored procedure that returns only the result you need.
- If SelectCommand does not return any rows, no tables are added to the DataSet, and no exception is raised.

See also

.NET Framework documentation for DdDataAdapter.fill() for the list of supported fill methods.

FillError event

Description

Returned when an error occurs during a fill operation.

Syntax

event FillErrorEventHandler **FillError**

Usage

The FillError event allows you to determine whether or not the Fill operation should continue after the error occurs. Examples of when the FillError event might occur are:

- The data being added to a DataSet cannot be converted to a common language runtime type without losing precision.
- The row being added contains data that violates a constraint that must be enforced on a DataColumn in the DataSet.

FillSchema method

Description

Adds DataTables to a DataSet and configures the schema to match the schema in the data source.

Syntax 1

DataTable[ ] **FillSchema(** DataSet *dataSet*, SchemaType *schemaType* **)**

Syntax 2

DataTable[ ] **FillSchema(** DataSet *dataSet*, SchemaType *schemaType*, string *srcTable* **)**

Syntax 3

DataTable **FillSchema(** DataTable *dataTable*, SchemaType *schemaType* **)**

Parameters

dataSet: A DataSet to fill with records and, optionally, schema.

schemaType: One of the SchemaType values that specify how to insert the schema.

srcTable: The name of the source table to use for table mapping.

dataTable: A DataTable.

Return Value

For Syntax 1 and 2, the return value is a reference to a collection of DataTable objects that were added to the DataSet. For Syntax 3, the return value is a reference to a DataTable.

See also

"Obtaining AseDataAdapter schema information" on page 61 and .NET Framework help at http://msdn.microsoft.com/.

GetFillParameters

Description

Parameters set by the user when executing a Select statement.

Syntax AseParameter[ ] **GetFillParameters( )**

Return value An array of IDataParameter objects that contains the parameters set by the user.

Implements IDataAdapter.GetFillParameters

InsertCommand property

Description An AseCommand that is executed against the database when an Update() is called that adds rows to the database to correspond to rows that were inserted in the DataSet.

Syntax AseCommand **InsertCommand**

Access Read-write

Usage

- When InsertCommand is assigned to an existing AseCommand object, the AseCommand is not cloned. The InsertCommand maintains a reference to the existing AseCommand.
- If this command returns rows, the rows can be added to the DataSet depending on how you set the UpdatedRowSource property of the AseCommand object.

See also "Update method" on page 133, "Inserting, updating, and deleting rows using the AseCommand object" on page 42, and "Inserting, updating, and deleting rows using the AseDataAdapter object" on page 51.

MissingMappingAction property

Description Determines the action to take when incoming data does not have a matching table or column.

Syntax MissingMappingAction **MissingMappingAction**

Access Read-write

Property Value One of the MissingMappingAction values. The default is Passthrough.

Implements IDataAdapter.MissingMappingAction

MissingSchemaAction property

Description Determines the action to take when the existing DataSet schema does not match incoming data.

Syntax MissingSchemaAction **MissingSchemaAction**

Access Read-write

Property Value One of the MissingSchemaAction values. The default is Add.

Implements IDataAdapter.MissingSchemaAction

RowUpdated event

Description Occurs during update after a command is executed against the data source. The attempt to update is made, so the event is initiated.

Syntax event AseRowUpdatedEventHandler **RowUpdated**

Usage The event handler receives an argument of type AseRowUpdatedEventArgs containing data related to this event. The following AseRowUpdatedEventArgs properties provide information specific to this event:

- Command
- Errors
- RecordsAffected
- Row
- StatementType
- Status
- TableMapping

For more information, see the .NET Framework documentation for OleDbDataAdapter.RowUpdated Event.

RowUpdating event

Description Occurs during update before a command is executed against the data source. The attempt to update is made, so the event is initiated.

Syntax event AseRowUpdatingEventHandler **RowUpdating**

Usage The event handler receives an argument of type AseRowUpdatingEventArgs containing data related to this event. The following AseRowUpdatingEventArgs properties provide information specific to this event:

- Command

- Errors
- Row
- StatementType
- Status
- TableMapping

For more information, see the .NET Framework documentation for OleDbDataAdapter.RowUpdating Event.

SelectCommand property

Description

An AseCommand that is used during Fill or FillSchema to obtain a result set from the database for copying into a DataSet.

Syntax

AseCommand **SelectCommand**

Access

Read-write

Usage

- When SelectCommand is assigned to a previously created AseCommand, the AseCommand is not cloned. The SelectCommand maintains a reference to the previously created AseCommand object.
- If the SelectCommand does not return any rows, no tables are added to the DataSet, and no exception is raised.
- The Select statement can also be specified in the AseDataAdapter constructor.

TableMappings property

Description

A collection that provides the master mapping between a source table and a DataTable.

Syntax

DataTableMappingCollection **TableMappings**

Access

Read-only

Usage

- The default value is an empty collection.
- When reconciling changes, the AseDataAdapter uses the DataTableMappingCollection collection to associate the column names used by the data source with the column names used by the DataSet.

Update method

Description
Updates the tables in a database with the changes made to the DataSet.

Syntax 1
int **Update(** DataSet *dataSet* **)**

Syntax 2
int **Update(** DataSet *dataSet*, string *srcTable* **)**

Syntax 3
int **Update(** DataTable *dataTable* **)**

Syntax 4
int **Update(** DataRow[ ] *dataRows* **)**

Parameters
dataSet: A DataSet to update with records and, optionally, schema.

srcTable: The name of the source table to use for table mapping.

dataTable: A DataTable to update with records and, optionally, schema.

dataRows: An array of DataRow objects used to update the data source.

Return Value
The number of rows successfully updated from the DataSet.

Usage
The Update is carried out using the InsertCommand, UpdateCommand, and DeleteCommand properties on each row in the data set that has been inserted, updated, or deleted.

See also
"DeleteCommand property" on page 127, "InsertCommand property" on page 130, "UpdateCommand property" on page 133, "Inserting, updating, and deleting rows using the AseDataAdapter object" on page 51, and .NET Framework documentation.

UpdateCommand property

Description
An AseCommand that is executed against the database when Update() is called to update rows in the database that correspond to updated rows in the DataSet.

Syntax
AseCommand **UpdateCommand**

Access
Read-write

Usage
- When UpdateCommand is assigned to a previously created AseCommand, the AseCommand is not cloned. The UpdateCommand maintains a reference to the previously created AseCommand object.
- If execution of this command returns rows, these rows can be merged with the DataSet depending on how you set the UpdatedRowSource property of the AseCommand object.

See also
"Update method" on page 133.

AseDataReader class

Description
A read-only, forward-only result set from a query or stored procedure.

Base classes
MarshalByRefObject

Implements
IDataReader, IDisposable, IDataRecord, IEnumerable, IListSource

Usage

- There is no constructor for AseDataReader. To get an AseDataReader object, execute an AseCommand:

```
AseCommand cmd = new AseCommand("Select emp_id from employee", conn );
AseDataReader reader = cmd.ExecuteReader();
```

- You can only move forward through an AseDataReader. If you need a more flexible object to manipulate results, use an AseDataAdapter.
- The AseDataReader retrieves rows as needed when you use cursors. For more information see the UseCursor parameter in the ConnectionString property in the "AseConnection constructors" on page 114.

See also
"ExecuteReader method" on page 104 and "Accessing and manipulating data" on page 36.

Close method

Description
Closes the AseDataReader class.

Syntax
void **Close( )**

Implements
IDataReader.Close

Usage
You must explicitly call the Close method when you are through using the AseDataReader.

Depth property

Description
A value indicating the depth of nesting for the current row. The outermost table has a depth of zero.

Syntax
int **Depth**

Access
Read-only

Property Value
The depth of nesting for the current row.

Implements
IDataReader.Depth

Dispose method

Description	Frees the resources associated with the object.
Syntax	void **Dispose( )**

FieldCount property

Description	The number of columns in the result set.
Syntax	int **FieldCount**
Access	Read-only
Property Value	When not positioned in a valid record set, 0; otherwise the number of columns in the current record. The default is -1.
Implements	IDataRecord.FieldCount
Usage	When not positioned in a valid record set, this property has a value of 0; otherwise, it is the number of columns in the current record. The default is -1.

GetBoolean method

Description	Gets the value of the specified column as a Boolean.
Syntax	bool **GetBoolean(** int *ordinal* **)**
Parameters	**ordinal:** An ordinal number indicating the column from which the value is obtained. The numbering is zero-based.
Return value	The value of the column.
Implements	IDataRecord.GetBoolean
Usage	No conversions are performed. Use this method to retrieve data from columns of type bit.

GetByte method

Description	Gets the value of the specified column as a Byte.
Syntax	byte **GetByte(** int *ordinal* **)**
Parameters	**ordinal:** An ordinal number indicating the column from which the value is obtained. The numbering is zero-based.

Return value: The value of the column.

Implements: IDataRecord.GetByte

Usage: No conversions are performed. Use this method to retrieve data from columns of type tinyint.

GetBytes method

Description: Reads a stream of bytes from the specified column offset into the buffer as an array, starting at the given buffer offset.

Syntax:

long **GetBytes(** int *ordinal*, long *dataIndex*, byte[ ] *buffer*, int *bufferIndex*, int *length* **)**

Parameters:

ordinal: An ordinal number indicating the column from which the value is obtained. The numbering is zero-based.

dataIndex: The index within the column value from which to read bytes.

buffer: An array in which to store the data.

bufferIndex: The index in the array to start copying data.

length: The maximum length to copy into the specified buffer.

Return value: The number of bytes read.

Implements: IDataRecord.GetBytes

Usage:

- GetBytes returns the number of available bytes in the field. In most cases, this is the exact length of the field. However, the number returned can be less than the true length of the field if GetBytes has already been used to obtain bytes from the field. This can be the case, for example, when the AseDataReader class is reading a large data structure into a buffer.
- If you pass a buffer that is a null reference (“Nothing” in Visual Basic), GetBytes returns the length of the field in bytes.
- No conversions are performed. Use this method to retrieve data from columns of type image, binary, timestamp, and varbinary.

GetChar method

Description: Gets the value of the specified column as a character.

Syntax:

char **GetChar(** int *ordinal* **)**

Parameters

ordinal: An ordinal number indicating the column from which the value is obtained. The numbering is zero-based.

Return value

Gets the value of the column.

Implements

IDataRecord.GetChar

Usage

- No conversions are performed. Use this method to retrieve data from columns of type tinyint, char(1), and varchar(1).
- Call AseDataReader.IsDBNull to check for null values before calling this method.

See also

"IsDBNull method" on page 145.

GetChars method

Description

Reads a stream of characters from the specified column offset into the buffer as an array starting at the given buffer offset.

Syntax

long **GetChars(** int *ordinal*, long *dataIndex*, char[ ] *buffer*, int *bufferIndex*, int *length* **)**

Parameters

ordinal: The zero-based column ordinal.

dataIndex: The index within the row from which to begin the read operation.

buffer: The buffer into which to copy data.

bufferIndex: The index for buffer to begin the read operation.

length: The number of characters to read.

Return value

The actual number of characters read.

Implements

IDataRecord.GetChars

Usage

GetChars returns the number of available characters in the field. In most cases this is the exact length of the field. However, the number returned can be less than the true length of the field if GetChars has already been used to obtain characters from the field. This can be the case, for example, when the AseDataReader is reading a large data structure into a buffer.

If you pass a buffer that is a null reference (Nothing in Visual Basic), GetChars returns the length of the field in characters.

No conversions are performed. No conversions are performed. Use this method to retrieve data from columns of type text, char, and varchar.

See also

"Handling BLOBs" on page 69.

GetDataTypeName method

Description	Gets the name of the source datatype.
Syntax	string **GetDataTypeName(** int *index* **)**
Parameters	**index:** The zero-based column ordinal.
Return Value	The name of the back-end datatype.
Implements	IDataRecord.GetDataTypeName

GetDateTime method

Description	Gets the value of the specified column as a DateTime object.
Syntax	DateTime **GetDateTime(** int *ordinal* **)**
Parameters	**ordinal:** The zero-based column ordinal.
Return Value	The value of the specified column.
Implements	IDataRecord.GetDateTime
Usage	• No conversions are performed, so the data retrieved must already be a DateTime object. • Call AseDataReader.IsDBNull to check for null values before calling this method. • Use this method if the corresponding Adaptive Server type in the database is: datetime, smalldatetime, date(Adaptive Server 12.5.1 or higher) and time (Adaptive Server 12.5.1 or higher).
See also	"IsDBNull method" on page 145.

GetDecimal method

Description	Gets the value of the specified column as a Decimal object.
Syntax	decimal **GetDecimal(** int *ordinal* **)**
Parameters	**ordinal:** An ordinal number indicating the column from which the value is obtained. The numbering is zero-based.
Return Value	The value of the specified column.
Implements	IDataRecord.GetDecimal

Usage

- No conversions are performed. Use this method to retrieve data from columns of type decimal, numeric, smallmoney and money.
- Call AseDataReader.IsDBNull to check for null values before calling this method.

See also

"IsDBNull method" on page 145.

GetDouble method

Description

Identifies the value of the specified column as a double-precision floating point number.

Syntax

double **GetDouble(** int *ordinal* **)**

Parameters

ordinal: An ordinal number indicating the column from which the value is obtained. The numbering is zero-based.

Return Value

The value of the specified column.

Implements

IDataRecord.GetDouble

Usage

- Call AseDataReader.IsDBNull to check for null values before calling this method.
- No conversions are performed, so the data retrieved must already be a double-precision floating point number.
- Use GetDouble for Adaptive Server type float with a precision greater than or equal to 16 and GetFloat for Adaptive Server types real and float with a precision less than 16.

See also

"IsDBNull method" on page 145.

GetFieldType method

Description

Identifies the type that is the datatype of the object.

Syntax

Type **GetFieldType(** int *index* **)**

Parameters

index: The zero-based column ordinal.

Return Value

The type that is the datatype of the object.

Implements

IDataRecord.GetFieldType

GetFloat method

Description
Identifies the value of the specified column as a single-precision floating point number.

Syntax
float **GetFloat(** int *ordinal* **)**

Parameters
ordinal: An ordinal number indicating the column from which the value is obtained. The numbering is zero-based.

Return Value
The value of the specified column.

Implements
IDataRecord.GetFloat

Usage
- No conversions are performed, so the data retrieved must already be a single-precision floating point number.
- Call AseDataReader.IsDBNull to check for null values before calling this method.
- Use GetFloat for Adaptive Server types real and float with a precision less than 16 and GetDouble for Adaptive Server type float with a precision greater than or equal to 16.

See also
"IsDBNull method" on page 145.

GetInt16 method

Description
Identifies the value of the specified column as a 16-bit signed integer.

Syntax
short **GetInt16(** int *ordinal* **)**

Parameters
ordinal: An ordinal number indicating the column from which the value is obtained. The numbering is zero-based.

Return Value
The value of the specified column.

Implements
IDataRecord.GetInt16

Usage
No conversions are performed. Use this method to retrieve data from columns of type smallint.

GetInt32 method

Description
Identifies the value of the specified column as a 32-bit signed integer.

Syntax
int **GetInt32(** int *ordinal* **)**

Parameters **ordinal:** An ordinal number indicating the column from which the value is obtained. The numbering is zero-based.

Return Value The value of the specified column.

Implements IDataRecord.GetInt32

Usage No conversions are performed. Use this method to retrieve data from columns of type int[eger].

GetList method

Description Implements IListSource.

Syntax IList GetList();

Implements IListSource

Usage Allows you to set the DataSource property of the .NET DataGrid object to the AseDataReader. The grid then uses this method to bind the results from the AseDataReader directly to its cells.

Typically, you would not directly use this method. As AseDataReader implements this method, it allows you to do the following:

```
using(AseCommand cmd = new AseCommand(select total_sales from titles
   where title_id = 'BU1032', conn)
{
   using(AseDataReader rdr = cmd.ExecuteReader())
   {
      MyGrid.DataSource = rdr;
   }
}
```

GetName method

Description Identifies the name of the specified column.

Syntax string **GetName(** int *index* **)**

Parameters **index:** The zero-based index of the column.

Return value The name of the specified column.

Implements IDataRecord.GetName

GetOrdinal method

Description
Identifies the column ordinal, given the column name.

Syntax
int **GetOrdinal(** string *name* **)**

Parameters
name: The column name.

Return Value
The zero-based column ordinal.

Implements
IDataRecord.GetOrdinal

Usage
GetOrdinal performs a case-sensitive lookup first. If it fails, a second search that is case-insensitive occurs.

GetOrdinal is Japanese kana-width insensitive.

Because ordinal-based lookups are more efficient than named lookups, it is inefficient to call GetOrdinal within a loop. Save time by calling GetOrdinal once and assigning the results to an integer variable for use within the loop.

GetSchemaTable method

Description
Returns a DataTable that describes the column metadata of the AseDataReader.

Syntax
DataTable **GetSchemaTable( )**

Return value
A DataTable that describes the column metadata.

Implements
IDataReader.GetSchemaTable

Usage
This method returns metadata about each column in the following order:

- ColumnName
- ColumnOrdinal
- ColumnSize
- DataType
- ProviderType
- IsLong
- AllowDBNull
- IsReadOnly
- IsRowVersion
- IsUnique

- IsKeyColumn
- IsAutoIncrement
- BaseSchemaName
- BaseCatalogName
- BaseTableName
- BaseColumnName

For more information about these columns, see the .NET Framework documentation for SqlDataReader.GetSchemaTable.

See also "Obtaining DataReader schema information" on page 48.

GetString method

Description Identifies the value of the specified column as a string.

Syntax

string **GetString(** int *ordinal* **)**

Parameters **ordinal:** An ordinal number indicating the column from which the value is obtained. The numbering is zero-based.

Return Value The value of the specified column.

Implements

IDataRecord.GetString

Usage No conversions are performed, so the data retrieved must already be a string.

Call AseDataReader.IsDBNull to check for null values before calling this method.

See also "IsDBNull method" on page 145.

GetUInt16 method

Description Identifies the value of the specified column as a 16-bit unsigned integer.

Syntax

UInt16 **GetUInt16(** int *ordinal* **)**

Parameters **ordinal** An ordinal number indicating the column from which the value is obtained. The numbering is zero-based.

Return Value The value of the specified column.

Usage

No conversions are performed, so the data retrieved must already be a 16-bit integer.

GetUInt32 method

Description

Identifies the value of the specified column as a 32-bit unsigned integer.

Syntax

UInt32 **GetUInt32(** int *ordinal* **)**

Parameters

ordinal An ordinal number indicating the column from which the value is obtained. The numbering is zero-based.

Return Value

The value of the specified column.

Usage

No conversions are performed, so the data retrieved must already be a 32-bit integer.

GetUInt64 method

Description

Identifies the value of the specified column as a 64-bit unsigned integer.

Syntax

UInt64 **GetUInt64(** int *ordinal* **)**

Parameters

ordinal An ordinal number indicating the column from which the value is obtained. The numbering is zero-based.

Return Value

The value of the specified column.

GetValue method

Description

Identifies the value of the column at the specified ordinal in its native format.

Syntax

object **GetValue(** int *ordinal* **)**

Parameters

ordinal: An ordinal number indicating the column from which the value is obtained. The numbering is zero-based.

Return Value

The value to return.

Implements

IDataRecord.GetValue

Usage

This method returns DBNull for null database columns.

GetValues method

Description
Identifies all the attribute columns in the current row.

Syntax
int **GetValues(** object[ ] *values* **)**

Parameters
values: An array of objects that holds an entire row of the result set.

Return value
The number of objects in the array.

Implements
IDataRecord GetValues

Usage
- For most applications, the GetValues method provides an efficient means for retrieving all columns, rather than retrieving each column individually.
- You can pass an Object array that contains fewer than the number of columns contained in the resulting row. Only the amount of data the Object array holds is copied to the array. You can also pass an Object array whose length is more than the number of columns contained in the resulting row.
- This method returns DBNull for null database columns.
- Gets the value of the column at the specified ordinal in its native format.

IsClosed property

Description
Returns “true” if the AseDataReader is closed; otherwise, it returns “false.”

Syntax
bool **IsClosed**

Access
Read-only

Property Value
“True” if the AseDataReader is closed; otherwise, “false.”

Implements
IDataReader.IsClosed

Usage
IsClosed and RecordsAffected are the only properties that you can call after the AseDataReader is closed.

IsDBNull method

Description
Identifies a value indicating whether the column contains null values.

Syntax
bool **IsDBNull(** int *ordinal* **)**

Parameters
ordinal: The zero-based column ordinal.

Return value
“True” if the specified column value is equivalent to DBNull; otherwise, “false.”

Implements	IDataRecord.IsDbNull
Usage	Call this method to check for null column values before calling the typed Get methods (for example, GetByte, GetChar, and so on) to avoid raising an exception.

Item property

Description	Identifies the value of a column in its native format. In C#, this property is the indexer for the AseDataReader class.
Syntax 1	object **this[** int *index* **]**
Syntax 2	object **this[** string *name* **]**
Parameters	**index:** The column ordinal. **name:** The column name.
Access	Read-only
Implements	IDataRecord.Item

NextResult method

Description	Advances the AseDataReader to the next result, when reading the results of batch SQL statements.
Syntax	bool **NextResult( )**
Return value	“True” if there are more result sets. Otherwise, “false”.
Implements	IDataReader.NextResult
Usage	Used to process multiple results, which can be generated by executing batch SQL statements. By default, the data reader is positioned on the first result.

Read method

Description	Reads the next row of the result set and moves the AseDataReader to that row.
Syntax	bool **Read( )**
Return value	Returns true if there are more rows. Otherwise, it returns “false”.

Implements IDataReader.Read

Usage The default position of the AseDataReader is prior to the first record. Therefore, you must call Read to begin accessing any data.

Example The following code fills a list box with the values in a single column of results:

```
while( reader.Read() )
{
   listResults.Items.Add( reader.GetValue( 0 ).ToString() );
}
listResults.EndUpdate();
reader.Close();
```

RecordsAffected property

Description The number of rows changed, inserted, or deleted by execution of the SQL statement.

Syntax int **RecordsAffected**

Access Read-only

Property Value The number of rows changed, inserted, or deleted. This is 0 if no rows were affected or the statement failed, or -1 for Select statements.

Implements IDataReader.RecordsAffected

Usage

- The number of rows changed, inserted, or deleted. The value is 0 if no rows were affected or the statement failed, and -1 for Select statements.
- The value of this property is cumulative. For example, if two records are inserted in batch mode, the value of RecordsAffected will be two.
- IsClosed and RecordsAffected are the only properties that you can call after the AseDataReader is closed.

AseDbType enum

Specifies Adaptive Server datatypes. See Table 5-2 for details on datatype mappings.

Members Binary

Bit

Char
Date
DateTime
Decimal
Double
Float
Integer
Image
LongVarChar
Money
Nchar
Numeric
NVarChar
Real
SmallDateTime
SmallMoney
Text
Time
TimeStamp
TinyInt
UniChar
UniVarChar
VarBinary
VarChar

Note Numeric and Decimal are limited to a precision of 26, rather than 38, the precision of Adaptive Server.

Datatype mapping

The following table shows the datatype mappings in Adaptive Server ADO.NET Data Provider.

Table 5-2: Adaptive Server ADO.NET datatype mappings

Adaptive Server database type	AseDbType enumerated	.NET Dbtype enumerated	.NET class name
binary	Binary	Binary	Byte[]

Adaptive Server database type	AseDbType enumerated	.NET Dbtype enumerated	.NET class name
bigint	BigInt	Int64	Int64
bit	Bit	Boolean	Boolean
char	Char	AnsiStringFixedLength	String
date	Date	Date	DateTime
datetime	DateTime	DateTime	DateTime
decimal	Decimal	Decimal	Decimal
double	Double	Double	Double
float(<16)	Real	Single	Single
float(>=16)	Double	Double	Double
image	Image	Binary	Byte[]
int[eger]	Integer	Int32	Int32
money	Money	Currency	Decimal
nchar	NChar	AnsiStringFixedLength	String
nvarchar	NVarChar	AnsiString	String
numeric	Numeric	VarNumeric	Decimal
real	Real	Single	Single
smalldatetime	SmallDateTime	DateTime	DateTime
smallint	SmallInt	Int16	Int16
smallmoney	SmallMoney	Currency	Decimal
text	Text	AnsiString	String
time	Time	Time	DateTime
timestamp	TimeStamp	Binary	Byte[]
tinyint	TinyInt	Byte	Byte
unichar	UniChar	StringFixedLength	String
unitext	Unitext	String	String
univarchar	UniVarChar	String	String
unsignedbigint	UnsignedBigint	UInt64	UInt64
unsignedint	UnsignedInt	UInt32	UInt32
unsignedsmallint	UnsignedSmallInt	UInt16	UInt16
varbinary	VarBinary	Binary	Byte[]
varchar	VarChar	AnsiString	String

AseError class

Description	Collects information relevant to a warning or error returned by the data source.
Base classes	Object
	There is no constructor for AseError.
See also	“Error handling” on page 78.

ErrorNumber property

Description	Number of the error message.
Syntax	public int **MessageNumber**
Access	Read-only

Message property

Description	A short description of the error.
Syntax	public string **Message**
Access	Read-only

SqlState property

Description	The Adaptive Server five-character SQL state following the ANSI SQL standard. If the error can be issued from more than one place, the five-character error code identifies the source of the error.
Syntax	public string **SqlState**
Access	Read-only

ToString method

Description	Identifies the complete text of the error message.
Syntax	public string **ToString( )**
Usage	The return value is a string is in the form “AseError:”, followed by the message. For example:

```
AseError:UserId or Password not valid.
```

Description The message state. Used as a modifier to the MsgNumber.

Syntax public int **State**

Description The severity of the message.

Syntax public int **Severity**

Description The name of the server that is sending the message.

Syntax public string **ServerName**

Description The name of the stored procedure or remote procedure call (RPC) in which the message occurred.

Syntax public string **ProcName**

Description The line number in the command batch or the stored procedure that has the error, if applicable.

Syntax public int **LineNum**

Description Associated with the extended message.

Syntax public int **Status**

Description The current state of any transactions that are active on this dialog.

Syntax public int **TranState**

Description The error message that comes from the Adaptive Server server.

Syntax bool **IsFromServer**

Usage The return value is “true” or “false.”

```
if (ex.Errors[0].IsInformation )
    MessageBox.Show("ASE has reported the following
error: " + ex.Errors[0].Message);
```

Description The error message that comes from Adaptive Server ADO.NET Data Provider.

Syntax bool **IsFromClient**

Usage The return value is “true” or “false.”

```
if (ex.Errors[0].IsInformation )
    MessageBox.Show("ASE has reported the following
error: " + ex.Errors[0].Message);
```

Description The message is considered an error.

Syntax bool **IsError**

Usage The return value is “true” or “false.”

```
if ( ! ex.Errors[0].IsInformation )
    MessageBox.Show("Error: " + ex.Errors[0].Message):
```

Description — The message is a warning that things might not be quite right.

Syntax — bool **IsWarning**

Usage — The return value is "true" or "false."

```
if ( ! ex.Errors[0].IsInformation )
    MessageBox.Show("Error: " + ex.Errors[0].Message):
```

Description — An informative message, providing information such as the active catalog has changed.

Syntax — bool **IsInformation**

Usage — The return value is "true" or "false."

```
if ( ! ex.Errors[0].IsInformation )
    MessageBox.Show("Error: " + ex.Errors[0].Message):
```

AseErrorCollection class

Description — Collects all errors generated by Adaptive Server ADO.NET Data Provider.

Base classes — Object

Implements — ICollection, IEnumerable

There is no constructor for AseErrorCollection. Typically, an AseErrorCollection is obtained from the AseException.Errors or InfoMessageArgs property.

See also — "Errors property" on page 153 and "Error handling" on page 78.

CopyTo method

Description — Copies the elements of the AseErrorCollection into an array, starting at the given index within the array.

Syntax — void **CopyTo(** Array *array*, int *index* **)**

Parameters — **array:** The array into which to copy the elements.

index: The starting index of the array.

Implements — ICollection.CopyTo

Count property

Description	The number of errors in the collection.
Syntax	int **Count**
Access	Read-only
Implements	ICollection.Count

Item property

Description	The error at the specified index.
Syntax	AseError **this[** int *index* **]**
Parameters	**index:** The zero-based index of the error to retrieve.
Property Value	An AseError that contains the error at the specified index.
Access	Read-only

AseException class

Description	The exception that is thrown when Adaptive Server returns a warning or error.
Base classes	SystemException

There is no constructor for AseException. Typically, an AseException object is declared in a catch. For example:

```
...
catch (AseException ex )
{
   MessageBox.Show(ex.Message, "Error" );
}
```

See also “Error handling” on page 78.

Errors property

Description	A collection of one or more AseError objects.
Syntax	AseErrorCollection **Errors**

Access: Read-only

Usage: The AseErrorCollection class always contains at least one instance of the AseError class.

Message property

Description: The text describing the error.

Syntax: string **Message**

Access: Read-only

Usage: This method returns the message for the first AseError.

AseFailoverException class

Description: The exception that is thrown when Adaptive Server successfully fails over to the secondary server configured in an HA cluster.

Base classes: AseException

There is no constructor for AseFailoverException. Typically, an AseFailoverException object is declared in a catch. For example:

```
...
catch( AseFailoverException ex )
{
   MessageBox.Show( ex.Message, "Warning!" );
}
```

See also: "AseException class" on page 153

Errors property

Description: The collection of warnings sent from the data source.

Syntax: AseErrorCollection **Errors**

Access: Read-only

Message property

Description	The full text of the error sent from the data source.
Syntax	string **Message**
Access	Read-only

ToString method

Description	Retrieves a string representation of the InfoMessage event.
Syntax	string **ToString( )**
Return value	A string representing the InfoMessage event.

AseInfoMessageEventArgs class

Description	The event arguments passed to the InfoMessage event handlers.
Base classes	EventArgs

Errors property

Description	A collection of the actual error objects returned by the server.
Syntax	AseErrorCollection **Errors**
Access	Read-only

Message property

Description	The error message.
Syntax	string **Message**
Access	Read-only

AseInfoMessageEventHandler delegate

Description: Represents the method that will handle the InfoMessage event of an AseConnection.

Syntax: void **AseInfoMessageEventHandler (** object *sender*, AseInfoMessageEventArgs *e* **)**

Parameters:

sender: The source of the event.

e: The AseInfoMessageEventArgs object that contains the event data.

AseParameter class

Description: Represents a parameter to an AseCommand and, optionally, its mapping to a DataSet column.

Base classes: MarshalByRefObject

Implements: IDbDataParameter, IDataParameter

AseParameter constructors

Syntax 1: **AseParameter( )**

Syntax 2: **AseParameter(** string *parameterName*, object *value* **)**

Syntax 3: **AseParameter(** string *parameterName*, AseDbType *dbType* **)**

Syntax 4: **AseParameter(** string *parameterName*, AseDbType *dbType*, int *size* **)**

Syntax 5: **AseParameter(** string *parameterName*, AseDbType *dbType*, int *size*, string *sourceColumn* **)**

Syntax 6: **AseParameter(** string *parameterName*, AseDbType *dbType*, int *size*, ParameterDirection *direction*, bool *isNullable*, byte *precision*, byte *scale*, string *sourceColumn*, DataRowVersion *sourceVersion*, object *value* **)**

Parameters:

value: An object that is the value of the parameter.

size: The length of the parameter.

sourceColumn: The name of the source column to map.

parameterName: The name of the parameter.

dbType: One of the AseDbType values.

direction: One of the ParameterDirection values.

isNullable: "True" if the value of the field can be null; otherwise, "false."

precision: The total number of digits to the left and right of the decimal point to which Value is resolved.

scale: The total number of decimal places to which Value is resolved.

sourceVersion: One of the DataRowVersion values.

AseDbType property

Description: The AseDbType of the parameter.

Syntax: AseDbType **AseDbType**

Access: Read-write

Usage:

- The AseDbType and DbType are linked. Therefore, setting the DbType changes the AseDbType to a supporting AseDbType.
- The value must be a member of the AseDbType enumerator.

DbType property

Description: The DbType of the parameter.

Syntax: DbType **DbType**

Access: Read-write

Usage:

- The AseDbType and DbType are linked. Therefore, setting the DbType changes the AseDbType to a supporting AseDbType.
- The value must be a member of the DbType enumerator.

Direction property

Description: A value indicating whether the parameter is input-only, output-only, bidirectional, or a stored procedure return value parameter.

Syntax: ParameterDirection **Direction**

Access: Read-write

Usage

If the ParameterDirection is output, and execution of the associated AseCommand does not return a value, the AseParameter contains a null value. After the last row from the last result set is read, Output, InputOut, and ReturnValue parameters are updated.

IsNullable property

Description

A value indicating whether the parameter accepts null values.

Syntax

bool **IsNullable**

Access

Read-write

Usage

This property is “true” if null values are accepted; otherwise, it is “false” (the default). Null values are handled using the DBNull class.

ParameterName property

Description

The name of the AseParameter.

Syntax

string **ParameterName**

Access

Read-write

Implements

IDataParameter.ParameterName

Usage

- Adaptive Server ADO.NET Data Provider uses positional parameters that are indicated with the *@name* parameter.
- The default is an empty string.
- Output parameters (whether prepared or not) must have a user-specified datatype.

Precision property

Description

The maximum number of digits used to represent the Value property.

Syntax

byte **Precision**

Access

Read-write

Implements

IDbDataParameter.Precision

Usage

- The value of this property is the maximum number of digits used to represent the Value property. The default value is 0, which indicates that Adaptive Server ADO.NET Data Provider sets the precision for the Value property.
- The Precision property is only used for decimal and numeric input parameters.

Scale property

Description

The number of decimal places to which Value is resolved.

Syntax

byte **Scale**

Access

Read-write

Implements

IDbDataParameter.Scale

Usage

The default is 0. The Scale property is only used for decimal and numeric input parameters.

Size property

Description

The maximum size, in bytes, of the data within the column.

Syntax

int **Size**

Access

Read-write

Implements

IDbDataParameter.Size

Usage

- The value of this property is the maximum size, in bytes, of the data within the column. The default value is inferred from the parameter value.
- The Size property is used for binary and string types.
- For variable length datatypes, the Size property describes the maximum amount of data to transmit to the server. For example, the Size property can be used to limit the amount of data sent to the server for a string value to the first 100 bytes.
- If not explicitly set, the size is inferred from the actual size of the specified parameter value. For fixed width datatypes, the value of Size is ignored. It can be retrieved for informational purposes, and returns the maximum amount of bytes Adaptive Server ADO.NET Data Provider uses when transmitting the value of the parameter to the server.

SourceColumn property

Description
The name of the source column mapped to the DataSet and used for loading or returning the value.

Syntax
string **SourceColumn**

Access
Read-write

Implements
IDbDataParameter.SourceColumn

Usage
When SourceColumn is set to anything other than an empty string, the value of the parameter is retrieved from the column with the SourceColumn name. If Direction is set to Input, the value is taken from the DataSet. If Direction is set to Output, the value is taken from the data source. A Direction of InputOutput is a combination of both.

SourceVersion property

Description
The DataRowVersion to use when loading Value.

Syntax
DataRowVersion **SourceVersion**

Access
Read-write

Implements
IDbDataParameter.SourceVersion

Usage
Used by UpdateCommand during an Update operation to determine whether the parameter value is set to Current or Original. This allows primary keys to be updated. This property is ignored by InsertCommand and DeleteCommand. This property is set to the version of the DataRow used by the Item property, or the GetChildRows method of the DataRow object.

ToString method

Description
Identifies a string containing the ParameterName.

Syntax
string **ToString( )**

Access
Read-write

Value property

Description
The value of the parameter.

Syntax
object **Value**

Access
Read-write

Implements
IDataParameter.Value

Usage

- For input parameters, the value is bound to the AseCommand that is sent to the server. For output and return value parameters, the value is set on completion of the AseCommand and after the AseDataReader is closed.
- When sending a null parameter value to the server, the user must specify DBNull, not null: the null value in the system is an empty object that has no value.
- If the application specifies the database type, the bound value is converted to that type when Adaptive Server ADO.NET Data Provider sends the data to the server. Adaptive Server ADO.NET Data Provider attempts to convert any type of value if it supports the IConvertible interface. Conversion errors can result if the specified type is not compatible with the value.
- Both the DbType and AseDbType properties can be inferred by setting the Value.
- The Value property is overwritten by Update.

AseParameterCollection class

Description
Represents all parameters to an AseCommand and, optionally, their mapping to a DataSet column.

Base classes
Object

Implements
ICollection, IEnumerable, IDataParameterCollection

Usage
There is no constructor for AseParameterCollection. You obtain an AseParameterCollection from the AseCommand.Parameters property.

See also
“Parameters property” on page 106.

Add method

Description
Adds an AseParameter to the AseCommand.

Syntax 1
public AseParameter **Add(** AseParameter *P* **)**

Syntax 2	public AseParameter **Add(** object *P***)**
Syntax 3	public AseParameter **Add(** string *name*, AseDbType *dataType* **)**
Syntax 4	public AseParameter **Add(** string *name*, object *value* **)**
Syntax 5	public AseParameter **Add(** string *name*, AseDbType *dataType*, AseParameter *size***)**
Syntax 6	public AseParameter **Add(** string *name*, AseDbType *dataType*, int *size*, string *sourceColumn***)**
Syntax 7	public AseParameter **Add(** string *parameterName*, AseDbType *dbType*, int *size*, ParameterDirection *direction*, Boolean *isNullable*, Byte *precision*, Byte *scale*, string *sourceColumn*, DataRowVersion *sourceVersion*, object *value***)**
Parameters	**value:** For Syntax 1 and 2, *value* is the AseParameter object to add to the collection. For Syntax 3, *value* is the value of the parameter to add to the connection. **parameterName:** The name of the parameter. **aseDbType:** One of the AseDbType values. **size:** The length of the column. **sourceColumn:** The name of the source column.
Return Value	Add inserts a parameter into the list of parameters help by the AseCommand. The return value is the new parameter added to the list.

Clear method

Description	Removes all items from the collection.
Syntax	void **Clear( )**
Implements	IList.Clear

Contains method

Description	Identifies a value indicating whether an AseParameter exists in the collection.
Syntax 1	bool **Contains(** object *value* **)**
Syntax 2	bool **Contains(** string *value* **)**
Parameters	**value:** The value of the AseParameter object to find. In syntax 2, this is the name.

Return value: "True" if the collection contains the AseParameter; otherwise, it is "false."

Implements:
- Syntax 1 implements IList.Contains.
- Syntax 2 implements IDataParameterCollection.Contains.

CopyTo method

Description: Copies AseParameter objects from the AseParameterCollection to the specified array.

Syntax: void **CopyTo(** array *array* int *index* **)**

Parameters:
array: The array into which to copy the AseParameter objects.
index: The starting index of the array.

Implements: ICollection.CopyTo

Count property

Description: Represents the number of AseParameter objects in the collection.

Syntax: int **Count**

Access: Read-only

Implements: ICollection.Count

IndexOf method

Description: Identifies the location of the AseParameter in the collection.

Syntax 1: int **IndexOf(** object *value* **)**

Syntax 2: int **IndexOf(** string *parameterName* **)**

Parameters:
value: The AseParameter object to locate.
parameterName: The name of the AseParameter object to locate.

Return Value: The zero-based location of the AseParameter in the collection.

Implements:
- Syntax 1 implements IList.IndexOf.
- Syntax 2 implements IDataParameterCollection.IndexOf.

Insert method

Description	Inserts an AseParameter in the collection at the specified index.
Syntax	void **Insert(** int *index* object *value* **)**
Parameters	**index:** The zero-based index where the parameter is to be inserted within the collection.
	value: The AseParameter to add to the collection.
Implements	IList.Insert

Item property

Description	The AseParameter at the specified index or name.
Syntax 1	AseParameter **this[** int *index* **]**
Syntax 2	AseParameter **this[** string *parameterName* **]**
Parameters	**index:** The zero-based index of the parameter to retrieve.
	parameterName: The name of the parameter to retrieve.
Property value	An AseParameter.
Access	Read-write
Usage	In C#, this property is the indexer for the AseParameterCollection class.

Remove method

Description	Removes the AseParameter that was passed into the method from the collection.
Syntax	void **Remove(** object *value* **)**
Parameters	**value:** The AseParameter object to remove from the collection.
Implements	IList.Remove

RemoveAt method

Description	Removes a parameter from the collection based on the parameter's index or name.
Syntax 1	void **RemoveAt(** int *index* **)**

Syntax 2

void **RemoveAt(** string *parameterName* **)**

Parameters

index: The zero-based index of the parameter to remove.

parameterName: The name of the AseParameter object to remove.

Implements

- Syntax 1 implements IList.RemoveAt.
- Syntax 2 implements IDataParameterCollection.RemoveAt.

AseRowUpdatedEventArgs class

Description

Provides data for the RowUpdated event.

Base classes

RowUpdatedEventArgs

AseRowUpdatedEventArgs constructors

Description

Initializes a new instance of the AseRowUpdatedEventArgs class.

Syntax

AseRowUpdatedEventArgs(DataRow *dataRow*, IDbCommand *command*, StatementType *statementType*, DataTableMapping *tableMapping* **)**

Parameters

dataRow: The DataRow sent through an Update.

command: The IDbCommand executed when Update is called.

statementType: One of the StatementType values that specifies the type of query executed.

tableMapping: The DataTableMapping sent through an Update.

Command property

Description

The AseCommand executed when Update is called.

Syntax

AseCommand **Command**

Access

Read-only

Errors property

Description	Any errors generated by Adaptive Server when the command was executed. Inherited from RowUpdatedEventArgs.
Syntax	Exception **Errors**
Property value	The errors generated by Adaptive Server when the Command was executed.
Access	Read-write

RecordsAffected property

Description	The number of rows changed, inserted, or deleted by execution of the SQL statement. Inherited from RowUpdatedEventArgs.
Syntax	int **RecordsAffected**
Property value	The number of rows changed, inserted, or deleted; 0 if no rows were affected or the statement failed; and -1 for Select statements.
Access	Read-only

Row property

Description	The DataRow sent through an Update. Inherited from RowUpdatedEventArgs.
Syntax	DataRow **Row**
Access	Read-only

StatementType property

Description	The type of the SQL statement that was executed. Inherited from RowUpdatedEventArgs.
Syntax	StatementType **StatementType**
Access	Read-only
Usage	StatementType can be one of Select, Insert, Update, or Delete.

Status property

Description	The UpdateStatus of the Command property. Inherited from RowUpdatedEventArgs.
Syntax	UpdateStatus **Status**
Property Value	One of the UpdateStatus values: Continue (the default), ErrorsOccurred, SkipAllRemainingRows, or SkipCurrentRow.
Access	Read-write

TableMapping property

Description	The DataTableMapping sent through an Update. Inherited from RowUpdatedEventArgs.
Syntax	DataTableMapping **TableMapping**
Access	Read-only

AseRowUpdatingEventArgs class

Description	Provides data for the RowUpdating event.
Base classes	RowUpdatingEventArgs

AseRowUpdatingEventArgs constructors

Description	Initializes a new instance of the AseRowUpdatingEventArgs class.
Syntax	**AseRowUpdatingEventArgs(** DataRow *row*, IDbCommand *command*, StatementType *statementType*, DataTableMapping *tableMapping* **)**
Parameters	**row:** The DataRow to update. **command:** The IDbCommand to execute during update. **statementType:** One of the StatementType values that specifies the type of query executed. **tableMapping:** The DataTableMapping sent through an Update.

Command property

Description	The AseCommand to execute when performing the Update.
Syntax	AseCommand **Command**
Access	Read-write

Errors property

Description	Any errors generated by Adaptive Server when the Command was executed. Inherited from RowUpdatingEventArgs.
Syntax	Exception **Errors**
Property value	The errors generated by Adaptive Server when the Command was executed.
Access	Read-write

Row property

Description	The DataRow sent through an Update. Inherited from RowUpdatingEventArgs.
Syntax	DataRow **Row**
Access	Read-only

StatementType property

Description	The type of the SQL statement that was executed. Inherited from RowUpdatingEventArgs.
Syntax	StatementType **StatementType**
Access	Read-only
Usage	StatementType can be Select, Insert, Update, or Delete.

Status property

Description	The UpdateStatus of the Command property. Inherited from RowUpdatingEventArgs.
Syntax	UpdateStatus **Status**

Property Value: One of the UpdateStatus values: Continue (the default), ErrorsOccurred, SkipAllRemainingRows, or SkipCurrentRow.

Access: Read-write

TableMapping property

Description: The DataTableMapping sent through an Update. Inherited from RowUpdatingEventArgs.

Syntax: DataTableMapping **TableMapping**

Access: Read-only

AseRowUpdatedEventHandler delegate

Description: Represents the method that will handle the RowUpdated event of an AseDataAdapter.

Syntax: void **AseRowUpdatedEventHandler (** object *sender*, AseRowUpdatedEventArgs *e* **)**

Parameters: **sender:** The source of the event.

e The AseRowUpdatedEventArgs that contains the event data.

AseRowUpdatingEventHandler delegate

Description: Represents the method that will handle the RowUpdating event of an AseDataAdapter.

Syntax: void **AseRowUpdatingEventHandler (** object *sender*, AseRowUpdatingEventArgs *e* **)**

Parameters: **sender:** The source of the event.

e The AseRowUpdatingEventArgs that contains the event data.

AseTransaction class

Description	Represents a SQL transaction.
Base classes	Object
Implements	IDbTransaction
Usage	• There is no constructor for AseTransaction. To obtain an AseTransaction object, use the AseConnection.BeginTransaction() method. • To associate a command with a transaction, use the AseCommand.Transaction property.
See also	"BeginTransaction method" on page 120, "Transaction processing" on page 76, and "Inserting, updating, and deleting rows using the AseCommand object" on page 42.

Commit method

Description	Commits the database transaction.
Syntax	void **Commit( )**
Implements	IDbTransaction.Commit

Connection property

Description	Identifies the AseConnection object associated with the transaction, or a null reference ("Nothing" in Visual Basic) if the transaction is no longer valid.
Syntax	AseConnection **Connection**
Access	Read-only
Usage	A single application can have multiple database connections, each with 0 or 1 transactions. This property allows you to determine the connection object associated with a particular transaction created by BeginTransaction.

IsolationLevel property

Description	Specifies the isolation level for this transaction.
Syntax	IsolationLevel *IsolationLevel*
Access	Read-only

Property Value: The IsolationLevel for this transaction. This can be ReadCommitted (the default), ReadUncommitted, RepeatableRead, or Serializable.

Implements: IDbTransaction.IsolationLevel

Rollback method

Description: Rolls the database transaction back.

Syntax: void **Rollback( )**

Implements: IDbTransaction.Rollback

Usage: Rollback is synchronous. You must first call BeginTransaction() and then call Rollback on the returned AseTransaction.

TraceEnterEventHandler delegate

Description: The method that handles the TraceEnter event.

Syntax: void **TraceEnterEventHandler(** AseConnection *connection*, Object *source*, string *method*, Object[] *parameters***)**

Parameters:

connection The connection the event occured on.

source The object that triggered the event.

method The method entered.

parameters The parameters to the method entered.

TraceExitEventHandler delegate

Description: The method that handles the TraceExit event.

Syntax: void **TraceExitEventHandler(** AseConnection *connection*, Object *source*, string *method*, Object[] *returnValue***)**

Parameters:

connection The connection the event occured on.

source The object that triggered the event.

method The method exited.

returnValue The return value of the method exited.

Index

A

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W