SYBASE®
An SAP® Company

Users Guide

Adaptive Server® Enterprise ADO.NET Data Provider 15.7 SP100

Microsoft Windows

DOCUMENT ID: DC20066-01-1570100-01
LAST REVISED: May 2013

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

Contents

Adaptive Server ADO.NET Data Provider 1
Requirements to Deploy Adaptive Server ADO.NET Data Provider 1
System Requirements 1
Adaptive Server ADO.NET Data Provider Assembly Deployment into Global Assembly Cache 2
Updating to a Newer Version of ADO.NET Data Provider 5
Common Language Runtime 5
Deploy Updates to the Data Provider 7
Sample Projects 7
Sample Applications 9
Tutorial: Use Simple Code Sample 9
Running Simple Code Sample in Visual Studio .NET 9
Running Simple Sample Project without Visual Studio 10
Understand Simple Sample Project 11
Tutorial: Use Table Viewer Code Sample 13
Running Table Viewer Code Sample in Visual Studio .NET 14
Running Table Viewer Sample Project without Visual Studio 15
Understand Table Viewer Sample Project 16
Tutorial: Use Advanced Code Sample 19
Running Advanced Code Sample in Visual Studio .NET 19
Running Advanced Sample Project without Visual Studio 20
Understand Advanced Sample Project 21

Develop Applications 25
Using Data Provider in a Visual Studio .NET Project . . . 25
Adding a Reference to Adaptive Server ADO.NET Data Provider in a Visual Studio .NET Project 25
Referencing the Adaptive Server ADO.NET Data Provider Classes in Code 26
Connecting to an Adaptive Server Database 26
Connection Pooling 28
Check the Connection State 29
Character Set 30
DDEX Provider for Adaptive Server 30
Connecting to Adaptive Server 31
Viewing Adaptive Server Objects 32
Supported Adaptive Server Objects 32
Enhanced DDEX Provider for Adaptive Server 33
Adaptive Server ADO.NET Data Provider Support for SSIS 34
Ways to Access and Manipulate Data 38
Use AseCommand to Retrieve and Manipulate Data 39
Use AseDataAdapter to Access and Manipulate Data 51
Obtaining Primary Key Values 65
BLOBs Handling 70
Obtaining Time Values 72
Converting a Time Value Using the GetDateTime Method 72
Stored Procedures 74
Executing a Stored Procedure 74
Alternate Way to Call a Stored Procedure 76
Transaction Processing 76
Isolation Level Setup for Transactions 77
Support for Transact-SQL Queries with COMPUTE clause 79

Error Handling 80
Performance Consideration 81
DbType.String vs. DbType.AnsiString 81
Supported Adaptive Server Cluster Edition features 83
Login Redirection 83
Connection Migration 83
Connection Failover 84
Distributed Transactions 84
Programming Using Enterprise Services 84
Directory Services 86
Microsecond Granularity for Time Data 87
Password Encryption 88
Password Expiration Handling 89
Secure Sockets Layer (SSL) 89
Failover in a High-Availability System 92
Kerberos Authentication 93
SECURECONNECTIONSTRING Property 96
Supported Microsoft ADO.NET 2.0, 3.0, 3.5 and 4.0 Features 97
Microsoft Enterprise Library Database Access Application Block for Adaptive Server 97
Microsoft ADO.NET Entity Framework and LINQ 98
Adaptive Server ADO.NET Data Provider API Reference 99
AseBulkCopy Class 99
AseBulkCopy Constructors 99
Close Method 100
Dispose Method 100
Finalize Method 100
WriteToServer Method 100
BatchSize Property 101
BulkCopyTimeout Property 101
ColumnMappings Property 101
DestinationTableName Property 101
NotifyAfter Property 102

AseRowsCopied Event 102
AseBulkCopyColumnMapping Class 102
AseBulkCopyColumnMapping Constructors 102
Equals Method 103
DestinationColumn Property 103
DestinationOrdinal Property 103
SourceColumn Property 103
SourceOrdinal property 104
AseBulkCopyColumnMappingCollection Class 104
AseBulkCopyColumnMappingCollection Constructor 104
Add Method 104
Contains Method 104
IndexOf Method 105
Insert Method 105
Validate Method 105
Remove Method 105
AseBulkCopyOptions Enumeration 106
AseClientFactory Class 106
AseClientFactory Constructors 106
Instance Field 107
CreateCommand Method 107
CreateCommandBuilder Method 107
CreateConnection Method 107
CreateConnectionStringBuilder Method 107
CreateDataAdapter Method 108
CreateDataSourceEnumerator Method 108
CreateParameter Method 108
CreatePermission Method 108
CanCreateDataSourceEnumerator Property 109
AseClientPermission Class 109
AseClientPermission Constructors 109
AseClientPermissionAttribute Class 110
AseClientPermissionAttribute Constructor 110
CreatePermission Method 110

AseCommand Class 110
AseCommand Constructors 111
Cancel Method 111
CommandText Property 112
CommandTimeout Property 112
CommandType Property 113
Connection Property 113
CreateParameter Method 114
ExecuteNonQuery Method 114
ExecuteReader Method 115
ExecuteScalar Method 115
ExecuteXmlReader Method 116
NamedParameters 116
Parameters Property 117
Prepare Method 117
Transaction Property 118
UpdatedRowSource Property 118
AseCommandBuilder Class 119
AseCommandBuilder Constructors 119
DataAdapter Property 119
DeleteCommand Property 120
DeriveParameters Method 120
Dispose Method 120
GetDeleteCommand Method 121
GetInsertCommand Method 121
GetUpdateCommand Method 122
InsertCommand Property 122
PessimisticUpdate Property 123
QuotePrefix Property 123
QuoteSuffix Property 124
RefreshSchema Method 124
SelectCommand Property 125
UpdateCommand Property 125
AseConnection Class 126
AseConnection Constructors 126

BeginTransaction Method 134
ChangeDatabase Method 135
Close Method 135
ConnectionString Property 136
ConnectionTimeout Property 137
CreateCommand Method 137
Database Property 138
InfoMessage Event 138
NamedParameters 138
Open Method 139
State Property 139
StateChange Event 139
TraceEnter, TraceExit Events 140
AseConnectionPool Class 140
Available Property 140
Size Property 141
AseConnectionPoolManager Class 141
AseConnectionPoolManager Constructor 141
GetConnectionPool Method 141
NumberOfOpenConnections Property 141
AseDataAdapter Class 142
AseDataAdapter Constructors 142
AcceptChangesDuringFill Property 143
ContinueUpdateOnError Property 143
DeleteCommand Property 144
Fill Method 144
FillError Event 145
FillSchema Method 146
GetFillParameters 146
InsertCommand Property 147
MissingMappingAction Property 147
MissingSchemaAction property 148
RowUpdated Event 148
RowUpdating Event 149
SelectCommand Property 149

TableMappings Property 150
Update Method 150
UpdateCommand Property 151
AseDataReader Class 151
Close Method 152
Depth property 152
Dispose Method 153
FieldCount Property 153
GetBoolean Method 153
GetByte Method 154
GetBytes Method 154
GetChar Method 155
GetChars Method 156
GetDataTypeName Method 157
GetDateTime Method 157
GetDecimal Method 158
GetDouble Method 158
GetFieldType Method 159
GetFloat Method 159
GetInt16 Method 160
GetInt32 Method 160
GetList Method 161
GetName Method 161
GetOrdinal Method 162
GetSchemaTable Method 162
GetString Method 163
GetUInt16 Method 164
GetUInt32 Method 164
GetUInt64 Method 164
GetValue Method 165
GetValues Method 165
IsClosed Property 166
IsDBNull Method 166
Item Property 167
NextResult Method 167

Read Method 168
RecordsAffected Property 168
AseDbType Enumeration 169
Adaptive Server ADO.NET Datatype Mappings 170
AseDecimal Structure 172
AseDecimal Constructors 172
AseDecimal Fields 172
CompareTo Method 173
Equality Operator 173
Equals Method 173
GetHashCode Method 174
GreaterThan Operator 174
GreaterThanOrEqual Operator 174
IsNegative property 174
IsNull Property 174
IsPositive Property 175
LessThan Operator 175
LessThanorEqual Operator 175
Parse Method 175
Sign Method 176
ToAseDecimal Method 176
ToString Method 176
AseError Class 176
ErrorNumber Property 177
Message Property 177
SqlState Property 177
ToString Method 177
AseErrorCollection Class 180
CopyTo Method 180
Count Property 181
Item Property 181
AseException Class 181
Errors Property 182
Message Property 182
AseFailoverException Class 182

Errors Property 183
Message Property 183
ToString Method 183
AseInfoMessageEventArgs Class 184
Errors Property 184
Message Property 184
AseInfoMessageEventHandler Delegate 184
AseParameter Class 185
AseParameter Constructors 185
AseDbType Property 186
DbType Property 186
Direction Property 187
IsNullable Property 187
ParameterName property 187
Precision Property 188
Scale Property 188
Size Property 189
SourceColumn Property 189
SourceVersion Property 190
ToString Method 190
Value Property 190
AseParameterCollection Class 191
Add Method 191
Clear Method 192
Contains Method 193
CopyTo Method 193
Count Property 193
IndexOf Method 194
Insert Method 194
Item Property 195
Remove Method 195
RemoveAt Method 195
AseRowsCopiedEventArgs Class 196
AseRowsCopiedEventArgs Constructor 196
Abort Property 196

RowCopied Property 197
AseRowsCopiedEventHandler Delegate 197
AseRowUpdatedEventArgs Class 197
AseRowUpdatedEventArgs Constructors 197
Command Property 198
Errors Property 198
RecordsAffected Property 198
Row Property 199
StatementType Property 199
Status Property 199
TableMapping Property 199
AseRowUpdatingEventArgs Class 200
AseRowUpdatingEventArgs Constructors 200
Command Property 200
Errors Property 200
Row Property 201
StatementType Property 201
Status Property 201
TableMapping Property 202
AseRowUpdatedEventHandler Delegate 202
AseRowUpdatingEventHandler Delegate 202
AseTransaction Class 202
Commit Method 203
Connection Property 203
IsolationLevel Property 204
Rollback Method 204
TraceEnterEventHandler Delegate 204
TraceExitEventHandler Delegate 205
Glossary 207
Index 209

Adaptive Server ADO.NET Data Provider

Adaptive Server® ADO.NET Data Provider is an ADO.NET provider for Adaptive Server Enterprise. It is supported on Adaptive Server versions 12.5.x, 15.0.x, 15.5 and 15.7.

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 interlanguage communication is provided by the Microsoft .NET framework.

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

- 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.

Requirements to Deploy Adaptive Server ADO.NET Data Provider

Review the requirements for deploying Adaptive Server ADO.NET Data Provider.

For a list of supported platforms, see *Sybase® Platform Certifications page* at *http://certification.sybase.com/ucr/search.do*.

System Requirements

Minimum system requirements for installing Adaptive Server ADO.NET Data Provider.

You must have the following installed on your machine:

- For development – NET Framework SDK 2.0 or later and Visual Studio .NET 2005 or later
- For deployment – NET Framework 2.0 or later

Required Files

Adaptive Server ADO.NET Data Provider consists of provider assembly files that are referenced by the client code.

- `Sybase.AdoNet2.AseClient.dll` – supports features of .NET 2.0, .NET 3.0, and .NET 3.5.

- `Sybase.AdoNet4.AseClient.dll` – supports features of .NET 4.0 and later.
  The 32-bit versions of these files are installed in the `C:\Sybase\DataAccess\ADONET\dll` directory and the 64-bit versions are installed in the `C:\Sybase\DataAccess64\ADONET\dll` directory.

Adaptive Server ADO.NET Data Provider Assembly Deployment into 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 you did not use the installation program, you must manually deploy the assembly. Do this by running **AseGacUtility** or **AseGacUtility4** (for .NET 4.0 and later) or by using the .NET Framework Configuration tool.

Deploying Assembly by Running AseGacUtility or AseGacUtility4

Run the **AseGacUtility** or **AseGacUtility4** utility to deploy assembly.

1. Go to the directory where **AseGacUtility** or **AseGacUtility4** is installed—by default `C:\Sybase\DataAccess\ADONET\dll` for Adaptive Server ADO.NET Data Provider, 32-bit and `C:\Sybase\DataAccess64\ADONET\dll` for Adaptive Server ADO.NET Data Provider, 64-bit.
2. **Run**

   ```
   AseGacUtility /i DLL_Name
   ```

   Or

   ```
   AseGacUtility4 /i DLL_Name
   ```

   where *DLL_Name* is the DLL you want to deploy in the GAC.

   For example, to deploy 32-bit version of `Sybase.AdoNet2.AseClient.dll`, run:

   ```
   AseGacUtility /i
   C:\Sybase\DataAccess\ADONET\dll\Sybase.AdoNet2.AseClient.dll
   ```

   Similarly, to deploy 64-bit version of `Sybase.AdoNet4.AseClient.dll`, run:

   ```
   AseGacUtility4 /i
   C:\Sybase\DataAccess64\ADONET\dll\Sybase.AdoNet4.AseClient.dll
   ```

Deploying Assembly Using the .NET Framework Configuration Tool

Use the .NET Framework configuration tool to deploy the assembly.

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.
4. In the Add an Assembly dialog box, find Adaptive Server ADO.NET Data Provider assembly in the installation directory and click **Open**.

   The default installation directories are `C:\Sybase\DataAccess\ADONET\dll` for Adaptive Server ADO.NET Data Provider, 32-bit and `C:\Sybase\DataAccess64\ADONET\dll` for Adaptive Server ADO.NET Data Provider, 64-bit. 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 and examine the list of assemblies in the cache.

Removing the Assembly from GAC

Remove the assembly from the GAC by running **AseGacUtility** or **AseGacUtility4** utility.

1. Go to the directory where AseGacUtility or AseGacUtility4 is installed—by default `C:\Sybase\DataAccess\ADONET\dll` for Adaptive Server ADO.NET Data Provider, 32-bit and `C:\Sybase\DataAccess64\ADONET\dll` for Adaptive Server ADO.NET Data Provider, 64-bit.
2. **Run**

   ```
   AseGacUtility /u DLL_Name
   ```

   Or

   ```
   AseGacUtility4 /i DLL_Name
   ```

   For example where *DLL_Name* is the DLL you want to remove from the GAC.

   For example, to remove 32-bit version of `Sybase.AdoNet2.AseClient.dll` from the GAC, run:

   ```
   AseGacUtility /u
   C:\Sybase\DataAccess\ADONET\dll\Sybase.AdoNet2.AseClient.dll
   ```

   For example, to remove 64-bit version of `Sybase.AdoNet4.AseClient.dll` from the GAC, run:

   ```
   AseGacUtility /u
   C:\Sybase\DataAccess64\ADONET\dll\Sybase.AdoNet4.AseClient.dll
   ```

Removing Assembly Using the .NET Framework Configuration Tool

Review the instructions to remove the assembly using .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.
4. Find `Sybase.AdoNet2.AseClient` or `Sybase.AdoNet4.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 to remove. Right-click and select **Delete**. Click **Yes** to confirm.
6. Check for the publisher policy files that correspond 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 the Windows installer. This happens due to some residual values in the registry. Contact Microsoft support for a resolution to this problem.

Deploying an Application Using Installation Program and GAC

Use 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. Copy your application-specific files, such as `exe` and `dlls`, to the system in the application-specific folder.

Deploying an Application Using GAC

Use 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*.

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 GAC

Review the instructions to deploy an application independent of 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

Updating ADO.NET Data Provider to a Newer Version. The updates to Adaptive Server ADO.NET Data Provider are delivered either through an EBF/ESD or maintenance releases.

To learn about issues updating to the newer version of Data Provider and Microsoft .NET concepts that pertain to updating, see *.NET Framework Deployment Guide* and *.NET Framework Developer's Guide* on the *Microsoft Developer Network at http://msdn.microsoft.com*:

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

- 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*
- Rebuild and redeploy your application against the new version of Adaptive Server ADO.NET Data Provider. Sybase recommends that you choose this step.

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 must 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.

Redirecting CLR Using Application Configuration Files

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.AdoNet2.AseClient"
                          publicKeyToken="95d94fac46c88e1e"
                          culture="neutral" />
        <bindingRedirect oldVersion="1.0.0.0-2.155.999.65535"
                         newVersion="2.155.1000.0"/>
      </dependentAssembly>
    </assemblyBinding>
  </runtime>
</configuration>
```

Refer to the *MSDN Library* for complete configuration file schema information.

Note: Each application needs its own separate configuration file.

Redirect CLR 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.

The publisher policy file directs all references from an older assembly version to the newly installed version. Unlike the application configuration files, publisher policy files must 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 at *http://msdn2.microsoft.com/en-us/default.aspx* 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. It deploys the new provider assembly and the publisher policy file in the GAC.

Deploy Updates to the Data Provider

Review the information to deploy updates to the Data Provider and any issues related to it.

Deploy Data Provider in to 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.

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.

Deploy Data Provider When Not in the GAC

If the Data Provider assembly is not installed in the GAC on this machine, copy the files that make up Data Provider into the application folder. To make the application use an updated version of Data Provider:

- 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.

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.

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.

See also

- *Sample Applications* on page 9

Sample Applications

You need access to an Adaptive Server with the pubs2 sample database installed to run the sample programs; also, you need either the Visual Studio .NET 2005 or the .NET Framework 2.0, 3.0, 3.5, or 4.0 installed.

The sample programs are located in these 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` for Adaptive Server ADO.NET Data Provider, 32-bit and `C:\Sybase\DataAccess64\ADONET\dll` for Adaptive Server ADO.NET Data Provider, 64-bit.

Tutorial: Use Simple Code Sample

The simple project has a number of features.

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

See also

- *Understand Simple Sample Project* on page 11

Running Simple Code Sample in Visual Studio .NET

Review the instructions to run simple code sample in Microsoft 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.Data.AseClient` and select **Remove**.
6. Add a reference to Adaptive Server ADO.NET Data Provider Assembly.
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.

See also

• *Adding a Reference to Adaptive Server ADO.NET Data Provider in a Visual Studio .NET Project* on page 25

Running Simple Sample Project without Visual Studio

Review the instructions to run simple sample project without Microsoft Visual Studio.

1. From a DOS prompt, go to the appropriate sample directory under `<install dir>\Samples`.
2. Add the directory with .NET Framework 2.0 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.

   The default installation directory is `C:\Sybase\DataAccess\ADONET\dll` for Adaptive Server ADO.NET Data Provider, 32-bit and `C:\Sybase\DataAccess64\ADONET\dll` for Adaptive Server ADO.NET Data Provider, 64-bit.
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.

Understand Simple Sample Project

Some key features of Adaptive Server ADO.NET Data Provider are illustrated 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.

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()
```

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()
```

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()
```

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
```

See also

- *AseConnection Class* on page 126
- *AseCommand Class* on page 110
- *AseDataReader Class* on page 151
- *AseException Class* on page 181

Tutorial: Use Table Viewer Code Sample

The 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

See also

- *Understand Table Viewer Sample Project* on page 16

Running Table Viewer Code Sample in Visual Studio .NET

Run the Table Viewer code sample project in Microsoft 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.Data.AseClient` and select **Remove**.
6. Add a reference to the Adaptive Server ADO.NET Data Provider Assembly.
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 corner of the window to terminate the application and disconnect from the sample database.

See also

• *Adding a Reference to Adaptive Server ADO.NET Data Provider in a Visual Studio .NET Project* on page 25

Running Table Viewer Sample Project without Visual Studio

Run the Table Viewer Sample Project without Microsoft 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 2.0 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.

   The default installation directory is `C:\Sybase\DataAccess\ADONET\dll` for Adaptive Server ADO.NET Data Provider, 32-bit and `C:\Sybase\DataAccess64\ADONET\dll` for Adaptive Server ADO.NET Data Provider, 64-bit.
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.

Understand Table Viewer Sample Project

Some key features of Adaptive Server ADO.NET Data Provider are illustrated 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
```

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()
```

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.

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
```

See also

- *AseConnection Class* on page 126
- *AseDataAdapter Class* on page 142
- *AseConnection Constructors* on page 126

Tutorial: Use Advanced Code Sample

The 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 these 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 Advanced Code Sample in Visual Studio .NET

Run the advanced code sample project in Microsoft 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.Data.AseClient` 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 Advanced Sample Project without Visual Studio

Run the advanced sample project without Microsoft 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 2.0 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.

   The default installation directory is `C:\Sybase\DataAccess\ADONET\dll` for Adaptive Server ADO.NET Data Provider, 32-bit and `C:\Sybase\DataAccess64\ADONET\dll` for Adaptive Server ADO.NET Data Provider, 64-bit.
4. Compile the sample program using the supplied build script `build.bat`.
5. 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.

Understand Advanced Sample Project

Some key features of Adaptive Server ADO.NET Data Provider are illustrated 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.

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
```

Develop Applications

Learn how to develop and deploy applications with the Adaptive Server ADO.NET Data Provider.

Using Data Provider in a Visual Studio .NET Project

Review the changes you must make to your Visual Studio .NET project, after you install Adaptive Server ADO.NET Data Provider.

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

See also

- *Requirements to Deploy Adaptive Server ADO.NET Data Provider* on page 1

Adding a Reference to Adaptive Server ADO.NET Data Provider in a Visual Studio .NET Project

Add a reference that informs Visual Studio .NET which assembly to include to find the code for Adaptive Server ADO.NET Data Provider.

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.AdoNet2.AseClient` or `Sybase.AdoNet4.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.AdoNet2.AseClient.dll` or `Sybase.AdoNet4.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, 32-bit and `C:\Sybase\DataAccess64\ADONET\dll` for Adaptive Server ADO.NET Data Provider, 64-bit.

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

Referencing the Adaptive Server ADO.NET Data Provider Classes in Code

Add a line to your source code to reference Adaptive Server ADO.NET Data Provider to be able to use it. You must add a different line for C# that for Visual Basic .NET.

1. Start Visual Studio .NET.
2. 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 an Adaptive Server Database

Write code to connect to an Adaptive Server database. Before you can carry out any operations, your application must connect to the 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 *Understand Table Viewer Sample Project*.

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';")
   ```

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.

See also

- *Understand Table Viewer Sample Project* on page 16
- *AseConnection Class* on page 126
- *ConnectionString Property* on page 136
- *AseConnection Constructors* on page 126

Connection Pooling

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.

Disable Connection Pooling

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

Check 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.

See also

- *State Property* on page 139

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 Client Charset or Server Default. Server Default is the default setting.

Usage

- When you choose ClientCharset as your character set, you 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, to set the negotiated character set to utf8:

  ```
  charset=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 these:
  - 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.

DDEX Provider for Adaptive Server

Data Designer Extensibility (DDEX) Provider for Adaptive Server enables Visual Studio components, such as Server Explorer, to interact with Adaptive Server and its objects through the Adaptive Server ADO.NET Data Provider.

With the DDEX Provider for Adaptive Server, you can:

- Connect and log in to Adaptive Server from Visual Studio.
- List Adaptive Server objects as hierarchy nodes in Visual Studio Server Explorer.
- Drag and drop Adaptive Server tables and views from Server Explorer onto data designers.

The DDEX Provider for Adaptive Server is compatible with Visual Studio 2005, 2008, and 2010.

Connecting to Adaptive Server

Add a database connection to Adaptive Server using the Visual Studio Server Explorer view.

Prerequisites

- Add the driver to the global assembly cache (GAC) if you have not yet installed SDK.
- If your application references an earlier version of the Adaptive Server ADO.NET Data Provider, run:
  - For Visual Studio 2005 and 2008:

    ```
    AseGacUtility -i
       policy.2.157.Sybase.AdoNet2.AseClient.dll
    ```

    ```
    AseGacUtility -i
       policy.2.157.Sybase.AdoNet4.AseClient.dll
    ```

  - For Visual Studio 2010:

    ```
    AseGacUtility4 -i
       policy.2.157.Sybase.AdoNet2.AseClient.dll
    ```

    ```
    AseGacUtility4 -i
       policy.4.157.Sybase.AdoNet4.AseClient.dll
    ```

- Run **AdoNetRegistrar** for Visual Studio 2005 and 2008, or **AdoNetRegistrar4** for Visual Studio 2010.

Task

1. In the Server Explorer view, right-click **Data Connections**.
2. Select **Add Connection**.
3. Select **Sybase ASE Database** as the data source and **.NET Framework Data Provider for Sybase ASE** as the data provider.
4. Enter the Adaptive Server additional connection properties.

   - Connection properties can be specified as a semicolon(;)-separated list.
   - Last connection property need not terminate with a semicolon(;).
   - Properties without a value are ignored.

   Currently, there are no warning or error messages to flag incorrect connection specifications.

Viewing Adaptive Server Objects

View Adaptive Server objects and their related information in Visual Studio Server Explorer.

Prerequisites

You need a valid connection to an active Adaptive Server to perform this task.

Task

1. Connect to Adaptive Server.
2. Expand the Adaptive Server object to explore.
3. Click an Adaptive Server object to view its property information.

Supported Adaptive Server Objects

DDEX Provider for Adaptive Server exposes Adaptive Server objects, which can be viewed and accessed in the Visual Studio Server Explorer.

Database Object	Properties
Check Constraint	Name, Catalog, Schema, Table, Constraint Column, Constraint Type
Column of Web Service as Table	Name, Catalog, Schema, Table, Default Value, Is Nullable, Ordinal, Length, Precision, Scale, System Type
Database	Name, Create Date, Dump Date
Datatype	Name
Default	Name, Catalog, Schema
Foreign Key	Name, Catalog, Schema, Table, Referenced Table, Referenced Table Catalog, Referenced Table Column, Referenced Table Schema
Index	Name, Catalog, Schema, Table, Referred Column
Instead-of Trigger	Name, Catalog, Schema, View
Primary Key	Name, Catalog, Schema, Table, Referred Column
Proxy Table	Name, Catalog, Schema
Proxy Table Column	Name, Catalog, Schema, Table, Default Value, Is Nullable, Ordinal, Length, Precision, Scale, System Type
Rule	Name, Catalog, Schema
Schema	Name
Stored Procedure	Name, Catalog, Schema

Database Object	Properties
Stored Procedure Parameter	Name, Catalog, Schema, Stored Procedure, Is Output, Ordinal, Length, Precision, Scale, System Type
Table	Name, Catalog, Schema, Type
Table Column	Name, Catalog, Schema, Table, Default Value, Is Nullable, Ordinal, Length, Precision, Scale, System Type
Trigger	Name, Catalog, Schema, Table
Unique Constraint	Name, Catalog, Schema, Table, Referred Column
User-defined Function	Name, Catalog, Schema
User-defined Function Parameter	Name, Catalog, Schema, User-defined Function, Is Output, Ordinal, Length, Precision, Scale, System Type
User-defined Type	Name, Catalog, Schema
View	Name, Catalog, Schema
View Column	Name, Catalog, Schema, View, Default Value, Is Nullable, Ordinal, Length, Precision, Scale, System Type
Web Services as Table	Name, Catalog, Schema, Method, Timeout, WSDL URI

Enhanced DDEX Provider for Adaptive Server

You can use Entity Framework to create data models with Adaptive Server databases. This enhancement is supported in Visual Studio 2008 SP1 and Visual Studio 2010.

Creating Adaptive Server Entity Data Model Using Entity Framework

Create data access classes using the Microsoft Entity Framework.

Prerequisites

Verify that you have a valid connection to an active Adaptive Server to perform this task.

Task

1. Create a new application project.
2. In the Solution Explorer window, right-click the project and select **Add** > **New Item**.
3. Select **Data** as the category and **ADO.NET Entity Data Model** as the template.
4. Enter a name for the entity data model and click **Add**.

   The Data Model wizard launches.
5. Choose **Generate** from database. Click **Next**.

6. Select an existing Adaptive Server connection or click **New Connection**. If you chose New Connection, enter the name of the connection settings. Click **Next**.
7. Select an Adaptive Server object. Click **Finish**.

Use the Entity Designer to modify the model. Data access classes are automatically generated by the Entity Framework when you save the model.

Adaptive Server ADO.NET Data Provider Support for SSIS

Adaptive Server ADO.NET Data Provider can be integrated into SQL Server Integration Services (SSIS), allowing for native access to ADO.NET Data Provider functions.

With the integration, you can use Adaptive Server as an:

- ADO.NET Connection Manager
- ADO.NET Source data flow component
- ADO.NET Destination data flow component

The enhanced Adaptive Server ADO.NET Data Provider supports SSIS 2008 and SSIS 2012. The SSIS support relies on the DDEX Provider for Adaptive Server which must be installed before SSIS can be used.

Setting up the Adaptive Server Connection

Configure the Adaptive Server connection.

Prerequisites

- Install the SDK that includes the Sybase ADO.NET Data Provider.
- Add the driver to the global assembly cache (GAC) if you have not yet installed SDK:

  ```
  AseGacUtility -i Sybase.AdoNet2.AseClient.dll
  ```

  ```
  AseGacUtility4 -i Sybase.AdoNet4.AseClient.dll
  ```

  Afterwards, run **AdoNetRegistrar**.

Task

1. On the Data Flow tab, right click on the ADO NET Source/Destination component you want to configure and select **Edit**.
2. Click the **New** button located next to Connection Manager.
3. In the Configure ADO.NET Connection Manager window, click **New**.
4. Select **Sybase Adaptive Server Enterprise Data Provider** from the listed providers.
5. Enter the appropriate connection properties.

   To use Adaptive Server ADO.NET with SSIS, set **QuotedIdentifier** to 1.
6. Click **OK**.

Note: By default, an ADO.NET Destination component batches the insert commands it performs. Currently, performing simple insert commands are faster than performing batch uploads to Adaptive Server through SSIS. To set the Destination component to perform simple insert commands, set the BatchSize property to 1.

SSIS Custom Data Flow Destination Component for Faster Data Transfers to Adaptive Server for SQL Server 2008

The Adaptive Server ADO.NET Data Provider distribution includes a SQL Server Integration Services (SSIS) Custom Data Flow Destination component, which performs faster data transfers into Adaptive Server destinations.

The faster data transfers use the Adaptive Server bulk-insert protocol supported by **AseBulkCopy** class. This component, named Sybase.AdoNet2.AseDestination.dll, is installed along with the Adaptive Server ADO.NET Data Provider and the assembly files in: `%SYBASE%\DataAccess\ADONET\` (32–bit systems) and `%SYBASE%\DataAccess64\ADONET\` (64–bit systems).

Configuring Adaptive Server ADO.NET Destination SSIS Component for SQLServer 2008

Configure Adaptive Server ADO.NET Destination SSIS component.

1. Copy the `Sybase.AdoNet2.AseDestination.dll` to `C:\Program Files\Microsoft SQL Server\100\DTS\PipelineComponents` and `C:\Program Files (x86)\Microsoft SQL Server\100\DTS\PipelineComponents`.
2. From any Microsoft SQL Server directory on your local drive, register the `Sybase.AdoNet2.AseDestination.dll` assembly using the AseGacUtility provided in the SDK installation.
3. Start SQL Server Business Intelligence Studio.
4. On the Toolbox tab, right-click Data Flow Destinations and select Choose Items. The Choose Toolbox Items window appears.
5. Select the SSIS Data Flow Items tab. Click Sybase Adaptive Server Enterprise ADO NET Destination, then click OK. Select **Toolbox > Data Flow Destinations** to see the Sybase Adaptive Server ADO NET Destination component.
6. To create an SSIS project, select **File > New > Project > Integration Services Project** menu. Create or drag and drop a Control Flow object from the Control Flow Items toolbox.
7. From the Data Flow Destinations and Data Flow Sources Toolbox tab, drag and drop Sybase Adaptive Server ADO NET Destination Component and ADO NET Source Component onto the Data Flow tab.
8. If a source or destination connection is not available in Connection Managers window, right-click in the Connection Managers window, and select New ADO.NET Connection. Select the already existing Data connection, or click New.

9. To create a new connection to the destination Adaptive Server, click New button in the Configure ADO.NET Connection Manager window, and then select Sybase Adaptive Server Enterprise Data Provider.
10. In the Connection Manager window, enter your connection properties.
11. To enable bulk insert, in the Additional Connection Props text box, enter: `enablebulkload=1`

Note: See **AseBulkCopy** in the *Adaptive Server Enterprise ADO.NET Data Provider Users Guide* for more details about utilizing bulk insert functionality.

12. Click OK.
13. For the ADO.NET Source in your Data Flow, setup the connection and data access mode. After you connect the data flow path from your ADO.NET Source, right-click Sybase Adaptive Server ADO NET Destination Component, and choose Show Advanced Edit.
14. From the Connection Manager tab, select ASE connection from the Connection Manager field. From the Component Properties tab, set the TableName property to the destination table name.
15. Select the Input Columns tab, and select the Name check box. This will select all the columns specified by the source table.
16. Click OK.

Note: The SSIS destination component for data transfers from SQL Server 2008 has been renamed from `Sybase.AdaptiveServerAdoNetDestination.dll` to `Sybase.AdoNet2.AseDestination.dll`.

The connection is established. See *Microsoft SSIS* documentation for more information about data transfer.

SSIS Custom Data Flow Destination Component for Faster Data Transfers to Adaptive Server for SQL Server 2012

The Adaptive Server ADO.NET Data Provider distribution includes a SQL Server Integration Services (SSIS) Custom Data Flow Destination component that is compatible with SQLServer 2012, which performs faster data transfer using bulk-insert protocol into Adaptive Server Enterprise.

The custom data flow destination component uses the Adaptive Server bulk-insert protocol supported by the `AseBulkCopy` class. This component, named `Sybase.AdoNet4.AseDestination.dll`, is installed along with the Adaptive Server ADO.NET Data Provider in `%SYBASE%\DataAccess\ADONET\dll` on 32-bit systems and `%SYBASE%\DataAccess64\ADONET\dll` on 64-bit systems.

Configuring the Adaptive Server ADO.NET Destination SSIS Component

The Adaptive Server ADO.NET Destination SSIS component performs faster data transfer into Adaptive Server destinations.

1. Copy `Sybase.AdoNet4.AseDestination.dll` to `C:\Program Files \Microsoft SQL Server\110\DTS\PipelineComponents` and `C: \Program Files (x86)\Microsoft SQL Server\110\DTS \PipelineComponents`.
2. From either of the Microsoft SQL Server directories on your local drive used in Step 1, register the `Sybase.AdoNet4.AseDestination.dll` using the `AseGacUtility4.exe` provided in the SDK installation.
3. To launch SQLServer 2012 Data Tools or SQL Server 2012 Data in Windows, select **Start > All Programs > Microsoft SQL Server 2012 > SQL Server Data Tools**.
4. Select **File > New > Project > Integration Services Project**.
   The Sybase Destination Component automatically appears in the SSIS Toolbox.
5. From the **Control Flow Items** toolbox drag and drop a Control Flow object.
6. Select the **Data Flow Destinations** tab, then select the **Data Flow Sources Toolbox** tab, then drag and drop **Sybase AdoNet4 ASE Destination** and **ADO NET Source Component** on to the Data Flow tab.
7. If there is no source or destination connection available in the Connection Managers window, right-click in the Connection Managers window, and select **New ADO.NET Connection**. If there is already an existing data connection, select it, or click **New**.
8. To create a new connection to the destination Adaptive Server, click **New** in the Configure ADO.NET Connection Manager window, then select **Sybase Adaptive Server Enterprise Data Provider**.
9. In the Connection Manager window, enter your connection properties.
10. To enable bulk insert, in the Additional Connection Props text box, enter:

    ```
    enablebulkload=1
    ```

    Note: See AseBulkCopy in the *Adaptive Server Enterprise ADO.NET Data Provider Users Guide* for more details about using bulk-insert.
11. Click **OK**.
12. For the ADO.NET source in your data flow, set up the connection and data access mode. After you connect the data flow path from your ADO.NET source, right-click **Sybase AdoNet4 ASE Destination**, and choose **Show Advanced Edit**.
13. From the **Connection Manager** tab, select the ASE connection from the Connection Manager field. From the **Component Properties** tab, set the TableName property to the destination table name.
14. Select the **Input Columns** tab, and select **Name**. This selects all the columns specified by the source table.

15. Click **OK** to establish the connection.

    See *Microsoft SSIS* documentation for more information about data transfers using SQL Server Integration Services.

Adaptive Server ADO.NET Data Provider Support for SSRS

The Adaptive Server ADO.NET Data Provider distribution includes a Microsoft SQL Server Reporting Services (SSRS) Custom Data Extensions component, which allows users to store credentials in the reporting server.

Adaptive Server SSRS component supports:

- Microsoft SQL Server 2008
- Microsoft SQL Server 2008 R2

This component, named Sybase.AdoNet2.AseReportingServices, is installed along with the Adaptive Server ADO.NET Data Provider in: `%SYBASE%\DataAccess\ADONET\dll` on 32-bit systems and `%SYBASE%\DataAccess64\ADONET\dll` on 64-bit systems.

Configuring the Adaptive Server ADO.NET SSRS Component

Configure the Adaptive Server ADO.NET SSRS component.

1. Copy `Sybase.AdoNet2.AseReportingServices.dll` to `C:\Program Files (x86)\Microsoft Visual Studio 9.0\Common7\IDE \PrivateAssemblies.`
2. Use a text editor to open the `RSReportDesigner.config` from `C:\Program Files (x86)\Microsoft Visual Studio 9.0\Common7\IDE \PrivateAssemblies`.
   - Enter the following below Data section:

     ```
     <Extension Name="Sybase"
     Type="Sybase.AdoNet2.AseReportingServices.SybaseClientConnecti
     onWrapper,Sybase.AdoNet2.AseReportingServices"/>
     ```
   - Enter the following below Designer section:

     ```
     <Extension Name="Sybase"
     Type="Microsoft.ReportingServices.QueryDesigners.GenericQueryD
     esigner,Microsoft.ReportingServices.QueryDesigners"/>
     ```
3. Save the `RSReportDesigner.config` file.

Ways to Access and Manipulate 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.

- **AseCommandObject** 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.

- **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.

See also

- *AseCommand Class* on page 110
- *AseDataReader Class* on page 151
- *Conflicts When Using the AseDataAdapter* on page 53
- *AseDataAdapter Class* on page 142

Use AseCommand to Retrieve and Manipulate Data

Review information to insert, update, or delete rows and retrieve data using the **AseDataReader**.

Get 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.

- **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.
- **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.

Instructions to use the Simple code sample included with Adaptive Server ADO.NET Data Provider.

See also

- *Understand Simple Sample Project* on page 11
- *ExecuteReader Method* on page 115
- *ExecuteScalar Method* on page 115
- *ExecuteXmlReader Method* on page 116

Issuing a Command that Returns a Complete Result Set

Issue 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* and *AseParameter Class*.

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()
   ```

See also

- *Stored Procedures* on page 74
- *AseParameter Class* on page 185

Issuing a Command that Returns Only One Value

Issue a command that returns a single complete result set.

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.

See also

- *AseDataReader Class* on page 151

Issuing a Command that Returns an XmlReader Object

Issue 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

Perform an Insert, Update, or Delete operation with AseCommand object using the `ExecuteNonQuery` function.

The **ExecuteNonQuery** function issues a command (SQL statement or stored procedure) that does not return a result set.

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.

See also

- *Transaction Processing* on page 76
- *ExecuteNonQuery Method* on page 114
- *Obtaining Primary Key Values* on page 65

Issuing a Command that Inserts a Row

Issue 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

Issue 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* and *AseParameter class*.

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()
```

See also

- *Stored Procedures* on page 74
- *AseParameter Class* on page 185

Issuing a Command that Deletes a Row

Issue 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()
```

Obtain 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.

See also

- *GetSchemaTable Method* on page 162

Obtaining Information About a Result Set Using the GetSchemaTable Method

Get 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
   ```

Use AseDataAdapter to Access and Manipulate Data

You can retrieve data, insert, update, or delete rows using the **AseDataAdapter**.

Get 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.

See also

- *AseDataAdapter Class* on page 142

Retrieving Data Using the AseDataAdapter Object

Fill a DataSet using the AseDataAdapter with the given example.

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")
   ```

Insert, Update, and Delete 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.

Conflicts When Using the AseDataAdapter

Your application logic must address some conflicts when using the AseDataAdapter.

- *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.
- *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.

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.

See also

- *Obtaining Primary Key Values* on page 65
- *Inserting, Updating, and Deleting Rows using the AseCommand Object* on page 44

Inserting Rows into a Table Using the AseDataAdapter

Insert 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 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

Use the AseDataAdapter object to update rows.

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

Use the AseDataAdapter object to delete rows from a table.

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

Get schema information about the result set in the DataSet using the **FillSchema** method along with the **AseDataAdapter**.

The **FillSchema** method returns the standard .NET `DataTable` object, which provides the names of all the columns in the result set.

See also

- *FillSchema Method* on page 146

Obtaining DataSet Schema Information Using the FillSchema Method

Use the `FillSchema` method to obtain dataset schema information.

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
   ```

Bulk-Load Support in Adaptive Server ADO.NET Data Provider

Adaptive Server ADO.NET Data Provider supports bulk-load interface for fast insertions of large sets of rows to Adaptive Server.

Setting the ENABLEBULKLOAD connection property allows **ASEBulkCopy** to invoke the bulk-load interface. Two types of bulk loading are supported:

- Array Inserts – use this type of bulk-loading within a single or multistatement transaction.
- Bulk Copy – this is supported only in single-statement transactions, and you must ensure that the **select into/bulkcopy** option on Adaptive Server is turned on.

  If the target table meets the criteria for high-speed version of **bulk copy**, Adaptive Server inserts the rows using this version of **bulk copy**.

  Note: Using the bulk-copy mode with the **select into/bulkcopy** option enabled affects the recoverability of the database. After the **bulk copy** operation is complete, the system administrator must dump the database to ensure its future recoverability.

Bulk-Load Option Usage

Use Case	Additional Consideration	Bulk-Load Option to Use	Note
Insertion of single or small number of rows.		None	
Insertion of large batch of rows.	The batch is part of a multistatement transaction.	Array Inserts	Rows are inserted faster than when bulk load is disabled.
	You cannot enable the Adaptive Server **select into** or **bulkcopy** option because of recoverability considerations.	Array Inserts	Rows are inserted faster than when bulk load is disabled.

Use Case	Additional Consideration	Bulk-Load Option to Use	Note
	The batch is a single transaction and the Adaptive Server **select into/bulkcopy** option is enabled.	Bulk Copy	Adaptive Server can use high-speed bulk copy, which is faster than array inserts. The performance of Bulk Copy is still slightly faster than Array Inserts even if high-speed bulk copy is not used.

See the *Adaptive Server Enterprise Utility Guide* for information about the implications of enabling **select into/bulkcopy** and the conditions under which high-speed or logged **bulk copy** is used.

Performance Considerations

Although this feature does not require special configuration on the server, a larger page size and network packet size significantly improves performance. Depending on the client memory, using larger batches also improves performance.

Supported ASEBulkCopy Options

ASEBulkCopy Options	**Supported Bulk-load Mode**
Default	Array Inserts, Bulk Copy, Off
KeepIdentity	Array Inserts, Bulk Copy, Off
KeepNulls	Array Inserts, Bulk Copy, Off
UseInternalTransaction	Array Inserts, Bulk Copy, Off
CheckConstraints	Off
FireTriggers	Off
TableLock	Not supported

Limitations

- Computed and encrypted columns are not supported. Also, triggers are ignored on tables selected for bulk-loading.
- The CheckConstraints, FireTriggers, and TableLock AseBulkCopy options are supported only as default values; these values are not supported when bulk-loading is disabled.

ENABLEBULKLOAD Connection Property

You can enable or disable bulk-load support using the `ENABLEBULKLOAD` connection property.

By setting the `ENABLEBULKLOAD` to:

- 0 – off mode, the default value.
- 1 – enables bulk-load using **array insert**.
- 2 – enables bulk-load using the **bulk copy** interface.
- 3 – enables bulk-load using the fast logged **bulk copy** interface.

Enabling Bulk Load Using the ADO.NET Connection String

Review the instructions to enable bulk-load using the ADO.NET connection string.

1. Use SQLDriverConnect to specify a connection string.
2. Set the **ENABLEBULKLOAD** connection string property to 0, 1, or 2, as appropriate.

   For example:

```
Data Source=server1;port=port1;UID=sa;PWD=;
Driver=AdaptiveServerEnterprise;
ENABLEBULKLOAD=1;
```

Obtaining Primary Key Values

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

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.

Use a table “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

Use the AseCommand object to insert a new row with an auto-incremented primary key.

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

Use the AseDataAdapter object to insert a new row with an auto-incremented primary key.

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()
   ```

BLOBs Handling

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.

See also

- *GetBytes Method* on page 154
- *GetChars Method* on page 156

Issuing a Command that Returns a String Using the GetChars Method

Use the `GetChars` method to issue a command that returns a string.

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

Use the `GetDateTime()` method if you want to fetch time values from Adaptive Server since the .NET Framework does not have a Time structure. Using this method returns the data as a .NET Framework DateTime object.

Converting a Time Value Using the GetDateTime Method

Use the `GetDateTime` method to convert a time value.

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()
```

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
```

See also

- *AseParameter Class* on page 185

Executing a Stored Procedure

Execute 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*.

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()
```

See also

- *AseParameter Class* on page 185

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”.

See also

- *Get Data Using the AseCommand Object* on page 39
- *Inserting, Updating, and Deleting Rows using the AseCommand Object* on page 44

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.

See also

- *AseTransaction Class* on page 202

Isolation Level Setup 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

Issue a command using an `AseTransaction` 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. 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. To make the changes permanent, use:

   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()
```

Support for Transact-SQL Queries with COMPUTE clause

Adaptive Server ADO.NET Data Provider supports Transact-SQL queries that include a **COMPUTE** clause.

A **COMPUTE** clause lets you include detail and summary results in a single **select** statement. The summary row follows the detail rows of a specific group, as shown here:

```
select type, price, advance from titles order by type compute
sum(price), sum(advance) by type
```

```
type          price      advance
------------  ---------  ----------
UNDECIDED     NULL       NULL
Compute Result:
----------------------- ------------------------
NULL                     NULL
type          price      advance
------------  ---------  ----------
business      2.99       10,125.00
business      11.95      5,000.00
business      19.99      5,000.00
business      19.99      5,000.00
Compute Result:
----------------------- ------------------------
54.92                    25,125.00
...
...

(24 rows affected)
```

When Adaptive Server ADO.NET Data Provider executes a **select** statement that includes a **COMPUTE** clause, the provider returns multiple result sets to the client. The number of result sets depends on the number of unique groupings available. Each group contains one result set for the detail rows and one result set for the summary. The client must process all result sets to fully process the rows returned; if it does not, only the detail rows of the first group of data are included in the first result set returned. See the *Adaptive Server Enterprise Transact-SQL Users Guide* for more information about the **COMPUTE** clause.

Error Handling

Your application must be designed to handle any errors that occur, including ADO.NET errors.

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

This 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
```

See also

- *Understand Simple Sample Project* on page 11
- *Understand Table Viewer Sample Project* on page 16
- *AseException Class* on page 181
- *AseError Class* on page 176

Performance Consideration

Review the tips on developing and deploying applications with the Adaptive Server ADO.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.

Supported Adaptive Server Cluster Edition features

Adaptive Server 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 allow 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

Login redirection is automatically enabled when a client application connects to a server that supports this feature.

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.

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 the 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

Set the HASession connection string property to 1 to enable Cluster Edition connection failover.

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.

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.

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

Connection Properties used in conjunction with Distributed Transaction support.

- Distributed Transaction Protocol (`DistributedTransactionProtocol`) – 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 – `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

Add the DSURL and Servername properties to the **ConnectString** to use directory services.

```
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?
sybaseServername=MANGO;
ldap://SYBLDAP1:389/dc=sybase,dc=com??one?sybaseServername=MANGO}
```

An example of DSURL follows:

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

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**.

Microsecond Granularity for Time Data

The Adaptive Server ADO.NET Data Provider provides microsecond-level precision for time data by supporting the SQL datatypes `bigdatetime` and `bigtime`.

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

- `bigdatetime` corresponds to the Adaptive Server `bigdatetime` datatype and indicates the number of microseconds that have passed since January 1, 0000

0:00:00.000000. The range of legal `bigdatetime` values is from January 1, 0001 00:00:00.000000 to December 31, 9999 23:59:59.999999.

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

Usage

- When connecting to Adaptive Server 15.5 and later, the Adaptive Server ADO.NET Data Provider transfers data using the `bigdatetime` and `bigtime` datatypes even if the receiving Adaptive Server columns are defined as `datetime` and `time`.

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

Password Encryption

The Adaptive Server ADO.NET Data Provider supports symmetrical and asymmetrical password encryption. You can use this feature to change the default behavior and encrypt your password, before they are sent over the network.

By default, the Adaptive Server ADO.NET Data Provider sends plain text passwords over the network to Adaptive Server for authentication.

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.

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

Enabling Password Encryption

Set the EncryptPassword connection property, which specifies whether the password is transmitted in encrypted format to enable password encryption.

The password is sent over the wire only after a login is negotiated; the password is first encrypted and then sent, after the password encryption is enabled. 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. Asymmetrical encryption requires additional processing time and may cause a slight delay in login time.

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;”;
```

Password Expiration Handling

Every company has a specific set of password policies for its database system. Depending on the policies, the password expires at a specific date and time.

Unless the password is reset, the Adaptive Server drivers connected to a database throw password expired errors and suggest that the user change the password using isql. The password change feature enables users to change their expired passwords without having to use another tool.

Adaptive Server ADO.NET Data Provider supports the **ChangePassword** method, which enables applications to change expired passwords without administrator intervention. For more information, search for ChangePassword method in the *Microsoft Developer Network*

Secure Sockets Layer (SSL)

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 web site at Internet Engineering Task Force *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 *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 different 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.

Validate 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 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*.

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

Add Encryption=ssl; TrustedFile=<trusted file> to the ConnectionString property to enable SSL for the Data Provider.

AseConnection then negotiates an SSL connection with the Adaptive 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*.

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
}
```

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).

Kerberos Process Overview

Review the Kerberos authentication process.

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.

Kerberos Requirements

You must configure Adaptive Server Enterprise to delegate authentication to Kerberos, to use it as an authentication system.

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

Enable Kerberos for the Adaptive Server ADO.NET Data Provider.
Add these instructions 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** method.

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;"
```

Obtaining an Initial Ticket From the Key Distribution Center

Generate an initial ticket called Ticket Granted Ticket (TGT) from the Key Distribution Center to use Kerberos authentication.

The procedure to obtain this ticket depends on the Kerberos libraries being used. For additional information, refer to the vendor documentation.

Explanation of Results

- **Ticket cache –** tells you which file contains your credentials cache.
- **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.

Generating TGTs for the MIT Kerberos Client Library

Review the instructions to generate TGTs for the MIT Kerberos Client Library.

1. Start the **kinit** utility at the command line:

   ```
   % kinit
   ```

2. Enter the **kinit** user name, such as your_name@YOUR.REALM.
3. Enter the password for your_name@YOUR.REALM, such as my_password. When you enter your password, the **kinit** utility submits a request to the Authentication Server for a Ticket Granting Ticket (TGT).

The password is used to compute a key, which in turn is used to decrypt part of the response. The response contains the confirmation of the request, as well as the session key. If you entered your password correctly, you now have a TGT.

4. 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
```

SECURECONNECTIONSTRING Property

SECURECONNECTIONSTRING is an Adaptive Server ADO.NET Data Provider connection property that removes the password property from the connection string.

This connection property ensures that the password is not exposed when the connection string is accessed using **AseConnection.ConnectionString**. Values:

- 0 – the default value; Adaptive Server ADO.NET Data Provider keeps the password in the connection string.
- 1 – Adaptive Server ADO.NET Data Provider removes the password from the connection string.

Supported Microsoft ADO.NET 2.0, 3.0, 3.5 and 4.0 Features

Adaptive Server ADO.NET Data Provider supports Microsoft ADO.NET 2.0, 3.0, 3.5, and 4.0 features.

- Provider factories
- Provider statistics
- Bulk update
- Bulk copy
- Asynchronous commands
- Extended pooling support to clear pools
- Common base classes
- Database metadata

For more information about these features, see the *Microsoft Developer Network* at *http://msdn.microsoft.com*.

Microsoft Enterprise Library Database Access Application Block for Adaptive Server

The Adaptive Server ADO.NET Data Provider 2.157 extends the Enterprise Library 4.1 Data Access Application Block (DAAB) to support Adaptive Server.

DAAB is a collection of classes that simplifies common database functions such as creating database instances and updating database records. DAAB also encapsulates database-specific features, which allows for a database-agnostic application design.

DAAB for Adaptive Server classes are supported in Microsoft .NET Framework 3.5 and Microsoft Visual Studio 2008. To use DAAB for Adaptive Server, you need to update Microsoft Enterprise Library 4.1.

The DAAB assembly adopted for Adaptive Server ADO.NET Data Provider is included with the installation as the `Sybase.EnterpriseLibrary.AseClient.dll` file. You can also install the DAAB source code adopted to build this assembly. The DAAB source code is installed in the `Sybase.EnterpriseLibrary.AseClient` directory when you choose to install the ADO.NET samples.

For information about the Enterprise Library Data Access Application Block, see the *Microsoft Developer Network* at *http://msdn.microsoft.com*.

Microsoft ADO.NET Entity Framework and LINQ

The Adaptive Server ADO.NET Data Provider supports the Visual Studio Language-Integrated Query (LINQ), the expanded Entity Data Model (EDM) canonical functions defined in Entity Framework 4.0, and the Microsoft ADO.NET Entity Framework including its LINQ-to-SQL component.

The following are unsupported due to Microsoft ADO.NET Entity Framework limitations:

- Use of the LINQ **Contains** extension method. **Contains** maps to the SQL **IN** clause.
- Creation of LINQ extension methods.
- Creation of associations between entity classes.

One of the advantages of ADO.NET Entity Framework and LINQ is that these allow you to work with a conceptual model of a relational storage schema, thus decreasing development and maintenance efforts for data-oriented applications. To use Microsoft ADO.NET Entity Framework and LINQ, reference `Sybase.AdoNet2.AseClient.dll` or `Sybase.AdoNet4.AseClient.dll`.

For information about the ADO.NET Entity Framework and LINQ, see the *Microsoft Developer Network* at *http://msdn.microsoft.com*.

Adaptive Server ADO.NET Data Provider API Reference

Review the information on 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*.

AseBulkCopy Class

Copies data from a data source to an Adaptive Server table.

Implements
IDisposable

AseBulkCopy Constructors

Initializes a new instance of the AseBulkCopy class.

Syntax 1

```
void AseBulkCopy( AseConnection connection)
```

Syntax 2

```
void AseBulkCopy( string connString )
```

Syntax 3

```
void AseBulkCopy( string connString, AseBulkCopyOptions copyOptions)
```

Syntax 4

```
void AseBulkCopy( AseConnection connection, AseBulkCopyOptions
copyOptions, AseTransaction externalTransaction )
```

Parameters

- **connection –** the Adaptive Server connection where the bulk copy operation is executed on.
- **connString –** the connection string to the connection where **AseBulkCopy** is executed on.

- **copyOptions –** the AseBulkCopy option.
- **externalTransaction –** the transaction to execute in the Adaptive Server connection.

See also

- *AseBulkCopyOptions Enumeration* on page 106

Close Method

Releases the resources used by the **AseBulkCopy** object.

Syntax

```
void Close()
```

Dispose Method

Releases the resources used by the **AseBulkCopy** object.

Syntax

```
void Dispose()
```

Implements

```
IDisposable.Dispose()
```

Finalize Method

Releases the resources used by the **AseBulkCopy** object.

Syntax

```
void Finalize()
```

WriteToServer Method

Writes data from the data source to the destination table.

Syntax 1

```
void WriteToServer( DataRow[ ] rows )
```

Syntax 2

```
void WriteToServer( DataTable table )
```

Syntax 3

```
void WriteToServer( IDataReader reader )
```

Syntax 4

```
void WriteToServer( DataTable table, DataRowState rowState)
```

Parameters

- **reader –** an **IDataReader**. Data from the interface is written to the data source.
- **rows –** rows of data in a **DataTable**. The data is written to the data source.
- **rowState –** specifies the state a data row must be in for it to be copied.
- **table –** a **DataTable**. Data from this table is written to the data source.

BatchSize Property

The number of rows in a batch.

Syntax

```
int BatchSize
```

Access

Read-write

BulkCopyTimeout Property

The number of seconds for the operation to execute before it times out.

Syntax

```
int BulkCopyTimeout
```

Access

Read-write

ColumnMappings Property

The collection of column mappings.

Syntax

```
AseBulkCopyColumnMappingCollection ColumnMappings
```

Access

Read-only

DestinationTableName Property

The name of the table to write data to.

Syntax

```
string DestinationTableName
```

Access

Read-write

NotifyAfter Property

The number of rows to process before triggering the **AseRowsCopied** event.

Syntax

```
int NotifyAfter
```

Access
Read-write

AseRowsCopied Event

Occurs every time the specified number of rows in **NotifyAfter** are processed.

Syntax

```
event AseRowsCopiedEventHandler AseRowsCopied
```

AseBulkCopyColumnMapping Class

Maps a column in the data source to a column in the destination table.

AseBulkCopyColumnMapping Constructors

Initializes a new instance of the AseBulkCopyColumnMapping class.

Syntax 1

```
void AseBulkCopyColumnMapping()
```

Syntax 2

```
void AseBulkCopyColumnMapping( int sourceInx, int dbInx )
```

Syntax 3

```
void AseBulkCopyColumnMapping( string sourceName, string dbName )
```

Parameters

- **dbInx –** the index of the column in the destination table.
- **sourceInx –** the index of the column in the data source.
- **dbName –** the name of the column in the destination table.
- **sourceName –** the name of the column in the data source.

Equals Method

Determines if two column mappings are the same.

Syntax

```
bool Equals( Object obj )
```

Parameter

obj – the object to compare with.

Return Value

True if the objects are the same; otherwise, false.

DestinationColumn Property

The name of the column to map to.

Syntax

```
string DestinationColumn
```

Access

Read-write

DestinationOrdinal Property

The index of the column to map to.

Syntax

```
int DestinationOrdinal
```

Access

Read-write

SourceColumn Property

The name of the column in the data source.

Syntax

```
string SourceColumn
```

Access

Read-write

SourceOrdinal property

The index of the column in the data source.

Syntax

```
int SourceOrdinal
```

Access
Read-write

AseBulkCopyColumnMappingCollection Class

A collection of column mappings.

AseBulkCopyColumnMappingCollection Constructor

Initializes a new instance of the AseBulkCopyColumnMappingCollection class.

Syntax

```
void AseBulkCopyColumnMappingCollection()
```

Add Method

Adds a new mapping to the collection.

Syntax

```
int Add( AseBulkCopyColumnMapping value )
```

Parameter
value – the column mapping to add.

Return Value
The index where the column mapping has been added.

Contains Method

Searches the collection for the specified mapping.

Syntax

```
bool Contains( AseBulkCopyColumnMapping value )
```

Parameter
value – the column mapping to search for.

Return Value
True if the collection contains the specified mapping; otherwise, false.

IndexOf Method

Retrieves the index of the specified mapping.

Syntax

```
int IndexOf( AseBulkCopyColumnMapping value )
```

Parameter
value – the column mapping to retrieve the index of.

Return Value
The index of the specified mapping.

Insert Method

Inserts the mapping at the specified index.

Syntax

```
void Insert( int index, AseBulkCopyColumnMapping value )
```

Parameters

- **index –** The position in the collection where the new mapping is inserted.
- **value –** The column mapping to add.

Validate Method

Performs custom processes to validate that the specified object can be added to the collection.

Syntax

```
void OnValidate( Object value )
```

Parameter
value – the object to validate.

Remove Method

Removes the mapping from the collection.

Syntax

```
void Remove( AseBulkCopyColumnMapping value )
```

Parameter
value – the column mapping to remove.

AseBulkCopyOptions Enumeration

Specifies the available behavior options for the AseBulkCopy class.

Syntax

```
enum AseBulkCopyOptions
```

Members

Member name	AseBulkCopy behavior
CheckConstraints	Checks constraints while during an insert operation.
Default	By default, all options are off.
FireTriggers	Informs the server to fire triggers during an insert operation.
KeepIdentity	Uses the identity values of the data source. If the KeepIdentity is not selected, Adaptive Server assigns the identity.
KeepNulls	Inserts null values. If KeepNulls is not selected, default values replace null values.
TableLock	Locks the entire table during the bulk copy operation.
UseInternalTransaction	If set, each batch of the bulk copy will be in a transaction.

AseClientFactory Class

A set of methods that you can use to create instances of the Adaptive Server ADO.NET Data Provider data source classes.

Base classes
DbProviderFactory

AseClientFactory Constructors

Initializes a new instance of the AseClientFactory class.

Syntax

```
void AseClientFactory()
```

Instance Field

An instance of AseClientFactory which you can use to retrieve strongly typed data objects.

CreateCommand Method

Returns a new instance of the Adaptive Server ADO.NET Data Provider class that implements the System.Data.Common.DbCommand class.

Syntax

```
DbCommand CreateCommand()
```

Return Value

A new instance of System.Data.Common.DbCommand.

CreateCommandBuilder Method

Returns a new instance of the Adaptive Server ADO.NET Data Provider class that implements the System.Data.Common.DbCommandBuilder class.

Syntax

```
DbCommandBuilder CreateCommandBuilder()
```

Return Value

A new instance of System.Data.Common.DbCommandBuilder.

CreateConnection Method

Returns a new instance of the Adaptive Server ADO.NET Data Provider class that implements the System.Data.Common.DbConnection class.

Syntax

```
DbConnection CreateConnection()
```

Return Value

A new instance of System.Data.Common.DbConnection.

CreateConnectionStringBuilder Method

Returns a new instance of the Adaptive Server ADO.NET Data Provider class that implements the System.Data.Common.DbConnectionStringBuilder class.

Syntax

```
DbConnectionStringBuilder CreateConnectionStringBuilder()
```

Return Value
A new instance of System.Data.Common.DbConnectionStringBuilder.

CreateDataAdapter Method

Returns a new instance of the Adaptive Server ADO.NET Data Provider class that implements the System.Data.Common.DbDataAdapter class.

Syntax

```
DbDataAdapter CreateDataAdapter()
```

Return Value
A new instance of System.Data.Common.DbDataAdapter.

CreateDataSourceEnumerator Method

Returns a new instance of the Adaptive Server ADO.NET Data Provider class that implements the System.Data.Common.DbDataSourceEnumerator class.

Syntax

```
DbDataSourceEnumerator CreateDataSourceEnumerator)
```

Return Value
A new instance of System.Data.Common.DbDataSourceEnumerator.

CreateParameter Method

Returns a new instance of the Adaptive Server ADO.NET Data Provider class that implements the System.Data.Common.DbParameter class.

Syntax

```
DbParameter CreateParameter()
```

Return Value
A new instance of System.Data.Common.DbParameter.

CreatePermission Method

Returns a new instance of the Adaptive Server ADO.NET Data Provider class that implements the System.Data.Common.CreatePermission class.

Syntax

```
CodeAccessPermission CreatePermission( PermissionState state )
```

Parameter
state – specifies whether a permission has access to resources.

Return Value
A new instance of System.Data.Common.CreatePermission.

CanCreateDataSourceEnumerator Property

Specifies whether the specific System.Data.Common.DbProviderFactory supports the System.Data.Common.DbDataSourceEnumerator class.

Syntax

```
bool CanCreateDataSourceEnumerator
```

Access
Read-only

Property Value
True if the instance of System.Data.Common.DbProviderFactory supports the System.Data.Common.DbDataSourceEnumerator class; otherwise, false.

AseClientPermission Class

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

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

Associates a security action with a custom security attribute.

Base Classes

```
DBDataPermissionAttribute
```

AseClientPermissionAttribute Constructor

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

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

Syntax

```
IPermission CreatePermission( )
```

AseCommand Class

Represents commands performed on the Adaptive Server and encapsulates either dynamic SQL statements or stored procedures.

It provides 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
```

Note: If you are calling a stored procedure that takes parameters, you must specify the parameters for the stored procedure.

See also

- *Use AseCommand to Retrieve and Manipulate Data* on page 39
- *Stored Procedures* on page 74
- *AseParameter Class* on page 185

AseCommand Constructors

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

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

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 111

CommandTimeout Property

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

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** – 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** – 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

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

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 185

ExecuteNonQuery Method

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

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 142
- *AseDataReader Class* on page 151

ExecuteScalar Method

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

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

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

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 191

Prepare Method

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

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

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

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 142

AseCommandBuilder Constructors

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

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

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

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

Frees the resources associated with the object.

Description

Syntax

```
void Dispose( )
```

GetDeleteCommand Method

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

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

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

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 150

PessimisticUpdate Property

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

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

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

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

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

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 150

AseConnection Class

Represents a connection to an Adaptive Server database.

Base Classes

```
Component
```

Implements

```
IDbConnection, IDisposable
```

See also

- *Connecting to an Adaptive Server Database* on page 26

AseConnection Constructors

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.

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';"
```

See also

- *Distributed Transactions* on page 84

Connection String Parameters

Review the table for connection string parameters.

Connection Properties	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

Connection Properties	Description	Required	Default Value
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 ServerDefault.
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
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 Properties	Description	Required	Default Value
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, 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

Connection Properties	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 orOleNative protocol when using distributed transactions. See Using Distributed Transactions, in Adaptive Server Advanced Features.	No	XA
EnableBulkLoad	Allows **ASEBulkCopy** to invoke the bulk-load interface. • 0 – off mode, the default value. • 1 – enables bulk-load using **array insert**. • 2 – enables bulk-load using the **bulk copy** interface. • 3 – enables bulk-load using the fast logged **bulk copy** interface.	No	0
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

Connection Properties	Description	Required	Default Value
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.
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

Connection Properties	Description	Required	Default Value
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
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.

Connection Properties	Description	Required	Default Value
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.
SupressRowFormat2	Forces Adaptive Server to send data using the TDS_ROWFMT byte sequence where possible instead of the TDS_ROWFMT2 byte sequence.	No	0
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, in 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

Connection Properties	Description	Required	Default Value
UseAseDecimal	Enables use of `AseDecimal` structure to retrieve numeric and decimal values. The `AseDecimal` structure can support a precision/scale of 78.	No	0
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
WindowsCharsetConverter	Allows the users to select which conversion library to use. • 0 – the Adaptive Server ADO.NET Data Provider will utilize Unilib library for character set conversions. • 1 – the Adaptive Server ADO.NET Data Provider will use the Microsoft Unicode conversion routines for character-set data conversion on Windows operating systems.	Yes, if user wants to use the Microsoft Unicode conversion routines for character-set data conversion on Windows operating systems	0

BeginTransaction Method

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 the *Adaptive Server Enterprise System Administration Guide* for information about inconsistencies.

See also
- *Transaction Processing* on page 76
- *Commit Method* on page 203
- *Rollback Method* on page 204

ChangeDatabase Method

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

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

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

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

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

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

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

Governs the default behavior of the `AseCommand` objects associated with this connection.

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

See also

- *NamedParameters* on page 116

Open Method

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

The current state of the connection.

Syntax

```
ConnectionState State
```

Access
Read-only

Default
Closed.

Implements

```
IDbConnection.State
```

See also

- *Check the Connection State* on page 29

StateChange Event

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

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.

Enable the ENABLETRACING connection property to trace Adaptive Server ADO.NET Data Provider activities. It is disabled by default to allow for better performance during normal execution where tracing is not needed. When this property is disabled, the **TraceEnter** and **TraceExit** events are not triggered, and tracing events are not executed. You can configure ENABLETRACING in the connection string using these values:

- True – triggers the **TraceEnter** and **TraceExit** events.
- False – the default value; Adaptive Server ADO.NET Data Provider ignores the **TraceEnter** and **TraceExit** events.

AseConnectionPool Class

Manages a pool of identical connections.

Available Property

The number of connections available in the pool.

Syntax

```
int Available
```

Access

Read-only

Size Property

The number of connections in the pool.

Syntax

```
int Size
```

Access
Read-only

AseConnectionPoolManager Class

Manages all connection pools.

AseConnectionPoolManager Constructor

Initializes a new instance of the AseConnectionPoolManager class.

Description

Syntax

```
void AseConnectionPoolManager()
```

GetConnectionPool Method

Retrieves the connection pool that manages the connections for the specified connection string.

Syntax

```
AseConnectionPool GetConnectionPool( string connectionString )
```

Parameter
connectionString – the connection string that identifies the pool to retrieve.

Return Value
The connection pool that manages *connectionString*.

NumberOfOpenConnections Property

The number of open connections in all of the connection pools.

Syntax

```
int NumberOfOpenConnections
```

Access
Read-only

AseDataAdapter Class

The link between the DataSet class and Adaptive Server.

Description
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

- *Use AseDataAdapter to Access and Manipulate Data* on page 51
- *Ways to Access and Manipulate Data* on page 38

AseDataAdapter Constructors

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

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

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

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

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

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

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 .NET Framework help at *http://msdn.microsoft.com/*.

See also

- *Obtaining AseDataAdapter Schema Information* on page 61

GetFillParameters

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

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

- *Insert, Update, and Delete Rows Using the AseDataAdapter Object* on page 52
- *Update Method* on page 150
- *Inserting, Updating, and Deleting Rows using the AseCommand Object* on page 44

MissingMappingAction Property

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

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

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

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

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

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

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 *.NET Framework documentation*.

See also

- *Insert, Update, and Delete Rows Using the AseDataAdapter Object* on page 52
- *DeleteCommand Property* on page 144
- *InsertCommand Property* on page 147
- *UpdateCommand Property* on page 151

UpdateCommand Property

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 150

AseDataReader Class

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*.

See also

- *Ways to Access and Manipulate Data* on page 38
- *ExecuteReader Method* on page 115

Close Method

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

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

Frees the resources associated with the object.

Syntax
```
void Dispose( )
```

FieldCount Property

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

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

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

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

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 166

GetChars Method

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

- *BLOBs Handling* on page 70

GetDataTypeName Method

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

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`and `time`.

See also

- *IsDBNull Method* on page 166

GetDecimal Method

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 166

GetDouble Method

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 166

GetFieldType Method

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

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 166

GetInt16 Method

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

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

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

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

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

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

- *Obtain DataReader Schema Information* on page 49

GetString Method

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 166

GetUInt16 Method

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

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

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

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

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

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

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

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

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

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

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 Enumeration

Specifies Adaptive Server datatypes.

See the Table 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.

Adaptive Server ADO.NET Datatype Mappings

The datatype mappings in Adaptive Server ADO.NET Data Provider.

Adaptive Server Database Type	AseDbType Enumerated	.NET Dbtype Enumerated	.NET class Name
binary	Binary	Binary	Byte[]
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

Adaptive Server Database Type	AseDbType Enumerated	.NET Dbtype Enumerated	.NET class Name
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

AseDecimal Structure

`AseDecimal` represents a decimal or numeric number with a maximum precision of 38. `AseDecimal` implements IComparable interface.

AseDecimal Constructors

Initializes a new instance of the `AseDecimal` structure.

Syntax 1

```
AseDecimal( AseDecimal asedecimal )
```

Syntax 2

```
AseDecimal( Decimal decimal )
```

Syntax 3

```
AseDecimal( Int32 precision, Int32 scale )
```

Syntax 4

```
AseDecimal( Int32 precision, Int32 scale, Byte[ ] value )
```

Parameters

- **asedecimal –** an `AseDecimal` structure to initialize the value of the new `AseDecimal`.
- **decimal –** a decimal structure to initialize the value of the new `AseDecimal`.
- **precision –** an Int32 value of the target precision.
- **scale –** an Int32 value of the target scale.
- **value –** the target value array in bytes.

AseDecimal Fields

Review the information on AseDecimal fields.

MAXLENGTH
Maximum DataLength is 33 bytes.

MAXOUTPUTPRECISION
Maximum output precision is 77.

MAXOUTPUTSCALE
Maximum output scale is 77.

MAXPRECISION
Maximum allowed precision is 38.

MAXSCALE
Maximum allowed scale is 38.

CompareTo Method

Compares the value of this `AseDecimal` with the target `AseDecimal` or object.

Syntax

```
CompareTo( AseDecimal )
```

Description
Compares the value of this `AseDecimal` with the *AseDecimal* value.

Syntax

```
CompareTo( Object )
```

Description
Compares the value of this `AseDecimal` with the value of *Object*.

Equality Operator

Returns a value indicating whether two instances of `AseDecimal` are equal.

Syntax

```
static bool operator ==( AseDecimal a, AseDecimal b )
```

Return Value
True if the values are equal.

Equals Method

Returns a value indicating whether this instance of `AseDecimal` and the value indicated by the object *value* are equal.

Syntax

```
bool Equals( Object value )
```

Parameters
value – the object being tested for equality.

GetHashCode Method

Returns a hashcode.

Syntax

```
override int GetHashCode( )
```

Return Value

An integer value representing the hashcode.

GreaterThan Operator

Returns a value indicating if *a* is greater than *b*.

Syntax

```
static bool operator >( AseDecimal a, AseDecimal b )
```

Return Value

True if *a* is greater than *b*.

GreaterThanOrEqual Operator

Returns a value indicating if *a* is greater than or equal to *b*.

Syntax

```
static bool operator >=( AseDecimal a, AseDecimal b )
```

Return Value

True if *a* is greater than or equal to *b*.

IsNegative property

Returns true if this `AseDecimal` is a negative value.

Syntax

```
bool IsNegative { get; }
```

IsNull Property

Returns true if this `AseDecimal` is a null value.

Syntax

```
bool  IsNull { get; }
```

IsPositive Property

Returns true if this `AseDecimal` is a positive value.

Syntax

```
bool  IsPositive { get; }
```

LessThan Operator

Returns a value indicating if *a* is less than *b*.

Syntax

```
static bool operator <( AseDecimal a, AseDecimal b )
```

Return Value

True if *a* is less than *b*.

LessThanorEqual Operator

Returns a value indicating if *a* is less than or equal to *b*.

Syntax

```
static bool operator <=( AseDecimal a, AseDecimal b )
```

Return Value

True if *a* is less than or equal to *b*.

Parse Method

Parses a string value to an `AseDecimal` value.

Syntax

```
AseDecimal Parse( string s )
```

Parameters

s – the string to be parsed into an AseDecimal value.

Return Value

An `AseDecimal` structure representing the parsed string.

Sign Method

Checks the sign of the `AseDecimal` value.

Syntax

```
int Sign( AseDecimal value )
```

Parameters

value – the AseDecimal structure whose design is being checked.

Return Value

Returns an integer `0` for null, `1` for positive, and `-1` for a negative values respectively.

ToAseDecimal Method

Converts this `AseDecimal` to new `AseDecimal` with specified precision and scale.

Syntax

```
AseDecimal ToAseDecimal( int outputPrecision, int outputScale )
```

Parameters

- **outputPrecision –** the target precision for the output
- **outputScale –** the target scale for the output.

Return Value

An `AseDecimal` structure with the specified precision and scale.

ToString Method

Returns a string representation of this `AseDecimal`.

Syntax

```
string ToString( )
```

Return Value

A string representation of this `AseDecimal`.

AseError Class

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 80

ErrorNumber Property

Number of the error message.

Syntax

```
public int MessageNumber
```

Access

Read-only

Message Property

A short description of the error.

Syntax

```
public string Message
```

Access

Read-only

SqlState Property

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

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

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

- *Error Handling* on page 80
- *Errors Property* on page 182

CopyTo Method

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

The number of errors in the collection.

Syntax

```
int Count
```

Access

Read-only

Implements

```
ICollection.Count
```

Item Property

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

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 80

Errors Property

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

The text describing the error.

Syntax

```
string Message
```

Access

Read-only

Usage

This method returns the message for the first `AseError`.

AseFailoverException Class

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 181

Errors Property

The collection of warnings sent from the data source.

Syntax

```
AseErrorCollection Errors
```

Access

Read-only

Message Property

The full text of the error sent from the data source.

Syntax

```
string Message
```

Access

Read-only

ToString Method

Retrieves a string representation of the **InfoMessage** event.

Syntax

```
string ToString( )
```

Return Value

A string representing the **InfoMessage** event.

AseInfoMessageEventArgs Class

The event arguments passed to the InfoMessage event handlers.

Base classes
EventArgs

Errors Property

A collection of the actual error objects returned by the server.

Syntax

```
AseErrorCollection Errors
```

Access
Read-only

Message Property

The error message.

Syntax

```
string Message
```

Access
Read-only

AseInfoMessageEventHandler Delegate

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

Represents a parameter to an **AseCommand** and, optionally, its mapping to a `DataSet` column.

Base classes

```
MarshalByRefObject
```

Implements

```
IDbDataParameter, IDataParameter
```

AseParameter Constructors

Review the information for AseParameter contructors.

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

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

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

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

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

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

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

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

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

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

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

Identifies a string containing the **ParameterName**.

Syntax

```
string ToString( )
```

Access
Read-write

Value Property

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

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 117

Add Method

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

Removes all items from the collection.

Syntax

```
void Clear( )
```

Implements

```
IList.Clear
```

Contains Method

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

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

Represents the number of `AseParameter` objects in the collection.

Syntax

```
int Count
```

Access

Read-only

Implements

```
ICollection.Count
```

IndexOf Method

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

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

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

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

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**.

AseRowsCopiedEventArgs Class

The argument data for the AseRowsCopiedEvent.

Base classes
EventArgs

AseRowsCopiedEventArgs Constructor

Initializes a new instance of the AseRowsCopiedEventArgs class.

Syntax

```
void AseRowsCopiedEventArgs( int num )
```

Parameter
num – the number of rows to copy.

Abort Property

Specifies whether to abort the bulk copy operation.

Syntax

```
bool Abort
```

Access
Read-write

RowCopied Property

The number of rows copied.

Syntax

```
int RowCopied
```

Access
Read-write

AseRowsCopiedEventHandler Delegate

The method that handles the AseRowsCopied event.

Syntax

```
void AseRowsCopiedEventHandler( Object sender, AseRowsCopiedEventArgs e )
```

Parameters

- **e –** the event arguments.
- **sender –** the object that triggered the event.

AseRowUpdatedEventArgs Class

Provides data for the RowUpdated event.

Base Classes

```
RowUpdatedEventArgs
```

AseRowUpdatedEventArgs Constructors

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

The **AseCommand** executed when **Update** is called.

Syntax

```
AseCommand Command
```

Access
Read-only

Errors Property

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

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

The DataRow sent through an **Update**. Inherited from **RowUpdatedEventArgs**.

Syntax

```
DataRow Row
```

Access
Read-only

StatementType Property

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

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

The DataTableMapping sent through an **Update**. Inherited from **RowUpdatedEventArgs**.

Syntax

```
DataTableMapping TableMapping
```

Access
Read-only

AseRowUpdatingEventArgs Class

Provides data for the RowUpdating event.

Base Classes

```
RowUpdatingEventArgs
```

AseRowUpdatingEventArgs Constructors

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

The **AseCommand** to execute when performing the **Update**.

Syntax

```
AseCommand Command
```

Access
Read-write

Errors Property

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

The DataRow sent through an Update. Inherited from **RowUpdatingEventArgs**.

Syntax

```
DataRow Row
```

Access
Read-only

StatementType Property

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

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

The DataTableMapping sent through an Update. Inherited from **RowUpdatingEventArgs**.

Syntax

```
DataTableMapping TableMapping
```

Access
Read-only

AseRowUpdatedEventHandler Delegate

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

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

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

- *Transaction Processing* on page 76
- *BeginTransaction Method* on page 134
- *Inserting, Updating, and Deleting Rows using the AseCommand Object* on page 44

Commit Method

Commits the database transaction.

Syntax

```
void Commit( )
```

Implements

```
IDbTransaction.Commit
```

Connection Property

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

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

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

The method that handles the TraceEnter event.

Syntax

```
void TraceEnterEventHandler( AseConnection connection,
   Object source, string method, Object[] parameters)
```

Parameters

- **connection –** the connection the event occurred on.
- **source –** the object that triggered the event.

- **method –** the method entered.
- **parameters –** the parameters to the method entered.

TraceExitEventHandler Delegate

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.

Glossary

Glossary of terms used in Adaptive Server® Enterprise ADO.NET Data Provider.

- **Adaptive Server Enterprise –** A server in Sybase's client/server architecture. Adaptive Server Enterprise manages multiple databases and multiple users, keeps track of the actual location of data on disks, maintains mapping of logical data description to physical data storage, and maintains data and procedure caches in memory.
- **ADO .Net (ActiveX Data Objects for .NET) –** ADO .Net (ActiveX Data Objects for .NET) is a set of computer software components that programmers can use to access data and data services. It is a part of the base class library that is included with the Microsoft .NET Framework. It is commonly used by programmers to access and modify data stored in relational database systems, though it can also access data in non-relational sources.
- **ADO.NET data provider –** An ADO.NET data provider is a software component enabling an ADO.NET consumer to interact with a data source.
- **BLOB –** Binary large object
- **CipherSuite –** A preferential list of key-exchange algorithms, hashing methods, and encryption methods used by SSL-enabled applications.
- **CLR (Common Language Runtime) –** The CLR is the virtual machine component of Microsoft's .NET framework and is responsible for managing the execution of .NET programs.
- **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.
- **connection migration –** Connection migration allows a server in a Cluster Edition environment to dynamically distribute load, and seamlessly migrate an existing client connection and its context to another server within the cluster.
- **connection pooling –** A connection pool is a cache of database connections maintained so that the connections can be reused when future requests to the database are required.
- **DAAB (Data Access Application Block) –** DAAB is a collection of classes that simplifies common database functions such as creating database instances and updating database records.
- **data provider –** Allows you to access data from server using any language supported by .Net.
- **data retrieval –** Data retrieval, in database management, involves extracting the wanted data from a database.
- **decryption –** The process of converting cipher text messages back to their plain text form.
- **distributed transaction –** A distributed transaction is an operations bundle, in which two or more network hosts are involved.

- **encryption –** The process that converts a plain text message to cipher text.
- **error handling –** Techniques available to Transact-SQL programmers on which to base code anddisplay errors and error messages.
- **GAC (Global Assembly Cache) –** The GAC is a machine-wide .NET assemblies cache for Microsoft's platform.
- **Kerberos –** Kerberos is an industry standard network authentication system that provides simple login authentication as well as mutual login authentication.
- **LDAP (Lightweight Directory Access Protocol) –** LDAP is an industry standard for accessing directory services.
- **LINQ (Language-Integrated Query) –** LINQ is a programming model that introduces queries as a first-class concept into any Microsoft .NET language.
- **object –** A passive entity that contains or receives information, and that cannot change the information it contains. In Adaptive Server, objects include rows, tables, databases, stored procedures, triggers, defaults, and views. See also database object.
- **ODBC (open database connectivity) –** The ODBC interface, defined by Microsoft Corporation, is a standard interface to database management systems in the Windows and Windows NT environments.
- **OLE DB provider –** An OLE DB provider is a software component enabling an OLE DB consumer to interact with a data source.
- **parameter –** A parameter is a special kind of variable, used in a subroutine to refer to arguments of data provided as input to the subroutine.
- **password expiration –** Every company has a specific set of password policies for its database system. Depending on the policies, the password expires at a specific date and time.
- **private key –** The secret key that must be kept secret in asymmetric algorithms.
- **public key –** The secret key that can be public in asymmetric algorithms.
- **schema –** A collection of objects associated with a particular schema name and schema authorization identifier. The objects can be tables, views, domains, constraints, assertions, privileges, and so on. A schema is created by a create schemastatement.
- **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.
- **transaction processing –** Transaction processing is information processing that is divided into individual, indivisible operations, called transactions. Each transaction must succeed or fail as a complete unit; it cannot remain in an intermediate state.
- **zero-downtime –** A migration or upgradation of a system or a database that is done less time .

Index

A

ADO.NET provider
    ASE ADO.NET Data Provider API 99
    connection pooling 28
    POOLING option 28
API reference
    ASE ADO.NET Data Provider API 99
ASE ADO.NET Data Provider
    adding a DLL reference in a C# project 25
    adding a DLL reference in a Visual Basic .NET project 25
    error handling 80
    running the sample projects 7
    system requirements 1
    using the code samples 9
ASE ADO.NET Data Provider API
    API reference 99
    AseCommand constructor 111
    AseCommandBuilder class 119
    AseCommandBuilder constructor 119
    AseConnection class 126
    AseConnection constructor 126
    AseDataAdapter class 142
    AseDataAdapter constructor 142
    AseDataReader class 151
    AseDbType enum 169
    AseDecimal constructors 172
    AseDecimal fields 172
    AseDecimal structure 172
    AseError class 176
    AseErrorCollection class 180
    AseException class 181
    AseInfoMessageEventArgs class 182
    AseInfoMessageEventHandler delegate 184
    AseParameter class 185
    AseParameter constructor 185
    AseParameterCollection class 191
    AsePermission class 109
    AsePermission constructor 109
    AsePermissionAttribute class 110
    AsePermissionAttribute constructor 110
    AseRowUpdatedEventArgs class 197
    AseRowUpdatedEventArgs constructor 197
    AseRowUpdatedEventHandler delegate 202
    AseRowUpdatingEventArgs class 200
    AseRowUpdatingEventHandler delegate 202
    AseTransaction class 202
    CreatePermission method 110
AseCommand class
    deleting data 44
    inserting data 44
    retrieving data 39
    updating data 44
    using 39
    using in a Visual Studio .NET project 12
AseCommand constructors
    ASE ADO.NET Data Provider API 111
AseCommandBuilder constructors
    ASE ADO.NET Data Provider API 119
AseConnection class
    ASE ADO.NET Data Provider API 126
AseConnection constructors
    ASE ADO.NET Data Provider API 126
AseConnection function
    using in a Visual Studio .NET project 16
AseDataAdapter class 51
    ASE ADO.NET Data Provider API 142
    retrieving data 51
    using in a Visual Studio .NET project 16
AseDataAdapter constructors
    ASE ADO.NET Data Provider API 142
AseDataReader class
    ASE ADO.NET Data Provider API 151
    using 39
    using in a Visual Studio .NET project 12
AseDbType enum
    ASE ADO.NET Data Provider API 169
    datatypes 169
AseDecimal constructors
    ASE ADO.NET Data Provider API 172
AseDecimal fields
    ASE ADO.NET Data Provider API 172
AseDecimal structure
    ASE ADO.NET Data Provider API 172
AseError class
    ASE ADO.NET Data Provider API 176
AseErrorCollection class
    ASE ADO.NET Data Provider API 180
AseException class
    ASE ADO.NET Data Provider API 181

AseInfoMessageEventArgs class
    ASE ADO.NET Data Provider API 182
AseInfoMessageEventHandler delegate
    ASE ADO.NET Data Provider API 184
AseParameter class
    ASE ADO.NET Data Provider API 185
AseParameter constructors
    ASE ADO.NET Data Provider API 185
AseParameterCollection class
    ASE ADO.NET Data Provider API 191
AsePermission class
    ASE ADO.NET Data Provider API 109
AsePermission constructors
    ASE ADO.NET Data Provider API 109
AsePermissionAttribute class
    ASE ADO.NET Data Provider API 110
AsePermissionAttribute constructors
    ASE ADO.NET Data Provider API 110
AseRowUpdatedEventArgs class
    ASE ADO.NET Data Provider API 197
AseRowUpdatedEventArgs constructors
    ASE ADO.NET Data Provider API 197
AseRowUpdatedEventHandler delegate
    ASE ADO.NET Data Provider API 202
AseRowUpdatingEventArgs class
    ASE ADO.NET Data Provider API 200
AseRowUpdatingEventHandler delegate
    ASE ADO.NET Data Provider API 202
AseTransaction class
    ASE ADO.NET Data Provider API 202
authentication 93

C

connection pooling
    ADO.NET provider 28
connection state
    ASE ADO.NET Data Provider 29
constructors
    AseCommand 111
    AseCommandBuilder class 119
    AseConnection constructor 126
    AseDataAdapter method 142
    AseParameter 185
    AsePermission constructor 109
    AsePermissionAttribute constructor 110
    AseRowUpdatedEventArgs constructor 197
CreatePermission method
    ASE ADO.NET Data Provider API 110

D

DataAdapter
    retrieving data 51
datatypes
    AseDbType enum 169
DDEX 30
delegates
    AseInfoMessageEventHandler delegate 184
    AseRowUpdatedEventHandler delegate 202
    AseRowUpdatingEventHandler delegate 202
developing applications with ASE ADO.NET Data Provider 25
directory services 86
DSURL 86

E

EncryptPassword 88
error handling
    ASE ADO.NET Data Provider 80
ExecuteReader method
    using 40
ExecuteScalar method
    using 42

F

failover 92

G

GAC
    deploying and configuring 7
GetSchemaTable method
    using 49
global assembly cache 2

H

HA 92
high availability 92

I

isolation levels
    setting for the AseTransaction object 77

K

Kerberos 93
    process overview 94
    requirements 94
kinit utility 95

L

LDAP 86

N

network authentication 93

O

objects
    ASE ADO.NET Data Provider API 99

P

password encryption 88
POOLING option
    ADO.NET provider 28
process overview
    Kerberos 94
publisher policy files 6

R

requirements
    Kerberos 94
response time
    AseDataAdapter 142
    AseDataReader 151

S

samples
    ASE ADO.NET Data Provider 9
Secure Sockets Layer 89
State property
    ASE ADO.NET Data Provider 29
Sybase.Data.AsaClient.DLL
    adding a reference to in a Visual Studio .NET project 25
system requirements
    ASE ADO.NET Data Provider 1

T

ToString method
    ASE ADO.NET Data Provider API 190
tutorials
    using the ASE ADO.NET Data Provider Table Viewer code sample 13

U

using the ASE ADO.NET Data Provider sample applications 9
using the Table Viewer code sample 13