Programming ADO SQL Server Applications

Microsoft® ActiveX® Data Objects (ADO) is a data access interface used to communicate with OLE DB-compliant data sources, such as Microsoft SQL Server[™] 2000. Data consumer applications can use ADO to connect to, retrieve, manipulate, and update data from an instance of SQL Server.

Architecturally, ADO is an application-level interface that uses OLE DB, a library of COM interfaces that enables universal access to diverse data sources. Because ADO uses OLE DB as its foundation, it benefits from the data access infrastructure that OLE DB provides; yet shields the application developer from the necessity of programming COM interfaces. Developers can use ADO for general-purpose access programs in business applications (Accounting, Human Resources, and Customer Management), and use OLE DB for tool, utility, or system-level development (development tools and database utilities).

The ADO topics emphasize the use of ADO 2.6 with SQL Server 2000, and are not intended as a general primer in using ADO. For more information about ADO 2.6, see the ADO documentation in the Microsoft Data Access Components (MDAC) SDK, located in the MSDN Library at <u>Microsoft Web</u> <u>site</u>. The ADO sections in MSDN Online contain ADO getting started topics and reference topics for ADO objects, collections, properties, and methods.

Data sources in SQL Server 2000 are suited for access through ADO. Because SQL Server is OLE DB-compliant, you can use ADO to develop client applications, service providers, Web applications, and business objects that access data in SQL Server 2000.

When programming ADO applications, consider:

- Which OLE DB provider to use.
- Which development environment to use.
- Additional data access requirements (for example, cursor types, transaction management, stored procedure usage, and so on).

A developer might also consider using Microsoft Remote Data Services (RDS). RDS is a Web-based technology that uses Microsoft Internet Information Services (IIS) and special ActiveX controls to bind data from an SQL data source to data controls on a Web page. RDS is integrated with ADO technology. For more information about RDS, see the RDS documentation in the Microsoft Data Access Components (MDAC) SDK, located in the Platform SDK in MSDN Online.

ADO can also be integrated with Microsoft® ActiveX® Data Objects (Multidimensional) (ADO MD), which you can use to browse a multidimensional schema, and query and retrieve the results of a cube; and Microsoft® ActiveX® Data Objects Extensions for Data Definition Language and Security (ADOX), which includes objects for schema creation and modification, and security.

Getting Started with ADO

Microsoft® SQL Server[™] 2000 applications can use ADO to connect to, retrieve, manipulate, and update data from an instance of SQL Server. These topics are discussed in Getting Started with ADO.

Торіс	Description
ADO Syntax Conventions	Describes text formatting used when explaining
	ADO syntax.
System Requirements for	Lists software required for using ADO to access
ADO	data in SQL Server 2000.
ADO and OLE DB	Describes the OLE DB providers that are
Provider Installation	installed with SQL Server 2000.
ADO File Locations	Lists the locations of all files that are required
	for ADO to communicate with SQL Server
	2000.
Upgrading the Catalog	Explains the process for updating catalog stored
Stored Procedures	procedures when using ADO with instances of
	SQL Server version 6.5 or earlier.
Using ADO in Different	Discusses using ADO with Microsoft Visual
<u>Development</u>	Basic [®] , Microsoft Visual C++ [®] , and using
<u>Environments</u>	ADO in Web-based applications.
Adding a Data Source	Describes how to add data sources for use with
	ADO.
Deleting a Data Source	Describes how to delete data sources.

ADO Syntax Conventions

ADO programming documentation uses the following conventions to distinguish elements of text.

Convention	Used for
UPPERCASE	Transact-SQL functions and statements, and C macro
	names.
courier new	Sample commands and program code.
italic	Function parameter names and information that the user or the application must provide.
bold	Function names, parameter keywords, and other syntax that must be typed exactly as shown.

System Requirements for ADO

To access data in Microsoft® SQL Server[™] 2000, you must have the following software installed:

- Microsoft OLE DB Provider for SQL Server (SQLOLEDB) or Microsoft OLE DB Provider for ODBC (MSDASQL).
- SQL Server 2000.
- Network software on the computers on which the driver and instance of SQL Server are installed (not required when connecting to a local desktop instance of SQL Server).

SQL Server

If you use the Microsoft OLE DB Provider for SQL Server (SQLOLEDB), or Microsoft OLE DB Provider for ODBC (MSDASQL) to access data in version 6.0, or 6.5 of SQL Server, you may need to install the catalog stored procedures. For more information, see Upgrading the Catalog Stored Procedures.

Network Software

Network software is required to connect the clients running SQLOLEDB or MSDASQL to the server on which an instance of SQL Server resides. To connect to a server running an instance of SQL Server, you can use the Microsoft Windows® 95, Microsoft Windows 98, Microsoft Windows NT® version 4.0, or Microsoft Windows 2000 operating system, or a compatible network such as Novell NetWare, or Banyan VINES. For information about the hardware and software required for each network, see the documentation for the network.

ADO and OLE DB Provider Installation

ADO clients that communicate with OLE DB need an OLE DB provider, a dynamic-link library that uses OLE DB interfaces and methods to query an SQL data source. For Microsoft® SQL Server[™] 2000, the following types of OLE DB providers can be used:

- Microsoft OLE DB Provider for SQL Server (SQLOLEDB), which maps OLE DB interfaces and methods over SQL Server data sources.
- Microsoft OLE DB Provider for ODBC (MSDASQL), which maps OLE DB interfaces and methods to ODBC APIs. OLE DB consumers connect to an instance of SQL Server using the SQL Server ODBC driver as an intermediary layer.

SQLOLEDB is installed with SQL Server 2000 and is recommended when developing new applications. MSDASQL is provided for backward compatibility only.

The ADO Connection and Error Handling sample application, used in some of the code examples in ADO topics, uses SQLOLEDB. Where pertinent, MSDASQL examples are also provided.

SQLOLEDB does not support the use of an ODBC DSN connection, but it does support the use of Microsoft Data Links. For more information about the use of connection properties for SQLOLEDB and MSDASQL, see <u>Connecting to a</u> <u>SQL Server Data Source</u>.

If you are running ADO code with SQL Server 2000 and an unexpected error occurs, check the provider properties. The error could be attributable to the way ADO interacts with different OLE DB providers.

ADO File Locations

All required ADO components are installed as part of either a Microsoft® SQL ServerTM 2000 server or client installation. You can develop ADO applications on either a client or a server.

ADO sample applications, which provide additional reference material for ADO application development, are included with SQL Server 2000. The ADO DLLs are installed automatically as part of SQL Server Setup in the C:\Program Files\Common Files\System\ADO directory.

Directory	File	Description
C:\Program Files\Common Files\System\ADO	ALL	Files implementing ADO objects.
C:\Program Files\Common Files\System\OLE DB	Sqloledb.dll	Dynamic-link library that implements the SQLOLEDB provider.
C:\Program Files\Microsoft SQL Server\80\Tools\Devtools\Include	Sqloledb.h	C/C++ header file used for developing SQLOLEDB consumers.
C:\Program Files\Common Files\System\OLE DB	Sqloledb.rll	SQLOLEDB resource file for developing Microsoft Visual Basic® applications.
C:\Program Files\Common Files\System\OLE DB	Msdasql.dll	Dynamic-link library that implements the MSDASQL provider.
C:\Program Files\Microsoft SQL Server\80\Tools\Devtools\Include	Msdasql.h	C/C++ header file used for developing MSDASQL consumers.

C:\Program Files\Microsoft SQL	ALL	Sample applications
Server\80\Tools\Devtools\Samples\Ado		that illustrate the use
		of ADO.

See Also

Overview of Installing SQL Server 2000

Upgrading the Catalog Stored Procedures

The Microsoft OLE DB Provider for SQL Server (SQLOLEDB) and Microsoft OLE DB Provider for ODBC (MSDASQL) can use a set of system stored procedures, known as catalog stored procedures, to obtain information from the SQL Server system catalog. SQL Server 2000 installs the catalog stored procedures automatically when you install or upgrade SQL Server. The Instcat.sql file includes updates to the catalog stored procedures. If the current version of SQLOLEDB or MSDASQL will be used against SQL Server version 6.5 or earlier, the SQL Server system administrator must upgrade the catalog stored procedures. Upgrading the catalog stored procedures does not affect the operation of existing SQL Server clients.

To upgrade the catalog stored procedures, the system administrator can run a script using the **osql** utility. To run **osql**, the computer must be installed as a client workstation for SQL Server. The system administrator should back up the **master** database before running Instcat.sql.

At a command prompt, use the **osql** utility to run the Instcat.sql script. For example:

C:> ISQL -Usa -Psa_password -Sserver_name -ilocation\Instcat.sql

Arguments

sa_password

System administrator password.

server_name

Name of the server on which an instance of SQL Server 2000 is installed.

location

Full path of the location of Instcat.sql. You can use Instcat.sql from an installed instance of SQL Server (the default location is C:\Program Files\Microsoft SQL Server\MSSQL\Install) or from the SQL Server 2000 compact disc (the default location is *D*:*platform* where *D* is the CD-ROM drive letter and *platform* is the appropriate server platform directory, such as

386).

The Instcat.sql script generates many messages. Most of these indicate how Transact-SQL statements issued by the script affected rows. These messages can be ignored, although the output should be scanned for messages that indicate an execution error. When Instcat.sql is run against SQL Server 6.0, the message generated about the object **sp_MS_upd_sysobj_category** not existing can be ignored. The last message should indicate that Instcat.sql completed successfully.

The Instcat.sql script fails when there is not enough space available in the **master** database to store the catalog stored procedures or to log the changes to existing procedures. If the Instcat.sql script fails, contact your system administrator.

The system administrator can also run Instcat.sql using SQL Query Analyzer.

Using ADO in Different Development Environments

The ADO object model is language neutral; it can be used in a variety of development environments. These include any of the Microsoft Visual languages (Microsoft® Visual Basic®, Microsoft Visual C++®, Microsoft Visual J++®), and Web development environments such as Microsoft Visual InterDevTM.

Visual Basic and ADO

With Microsoft® Visual Basic®, the ADO object model is integrated into the development environment. This allows you to use features such as drop-down lists of ADO properties and methods as you enter code, and internally, high-level access to OLE DB functionality.

Visual Basic version 6.0 includes:

- The ADO Data Control and other ADO/OLE DB capable data bound controls.
- The Data Environment Designer, an interactive graphical tool that allows for the building of ADO connections and commands. It provides a programmatic interface to the data access objects in a project.
- Dynamic data binding, which allows the run-time setting of a DataSource property of a data consumer, such as a DataGrid control, to a data source, such as the ADO Data Control.

To use ADO to access SQL Server 2000 data in a Visual Basic application

- 1. Reference ADO from your Visual Basic Project.
- 2. Set the Provider property of the **Connection** object by specifying **Sqloledb**.

To reference ADO from a Visual Basic project

- 1. In Visual Basic, on the **Project** menu, click **References**.
- 2. Select **Microsoft ActiveX Data Objects 2.6 Library**. Verify that at least the following libraries are also selected:
 - Visual Basic for Applications

- Visual Basic runtime objects and procedures
- Visual Basic objects and procedures
- OLE Automation

The library for ADO is msado15.dll and the program ID (ProgID) is ADODB.

For more information about the use of connection properties for SQLOLEDB, see <u>Connecting to a SQL Server Data Source</u>.

For more information about Visual Basic, see the MSDN Library at <u>Microsoft</u> <u>Web site</u>.

Visual C++ and ADO

Using Microsoft® Visual C++® with ADO allows you to write data access applications for Microsoft SQL Server[™] 2000. When developing a SQL Server application, you can:

- Use the #import Compiler COM directive to import the Msado15.dll before using ADO. The directive generates header files containing typedef declarations, smart pointers for interfaces, and enumerated constants. Each interface is encapsulated, or wrapped, in a class. This is the recommended way to program ADO using Visual C++.
- Use the **IADORecordBinding** interface (also referred to as ADO Visual C++ Extensions), which supports retrieving data into native C/C++ data types without going through a VARIANT data type. It also provides preprocessor macros when using the interface. The interface has methods to associate ADO Recordset fields with C/C++ variables, to add new rows, and to perform updates. This method of programming ADO using Visual C++ is recommended for backward compatibility only.

Visual Studio version 6.0 includes the ADO Data Control and other databound controls that you can use to design Microsoft Win32® applications that use ADO.

The Component Gallery contains the ADO Data Bound Dialog Wizard, which guides you through the process of creating a Microsoft Foundation Class Library (MFC) data bound dialog box with ADO. The controls of the dialog box bind to the fields of a recordset. Using the wizard, you can automatically generate all of the resources, classes, and Component Object Model (COM) initialization code necessary to build a data bound dialog box and add it to your project.

For more information about using Visual C++ with ADO, see the MSDN Library at <u>Microsoft Web site</u>.

Web-Based Applications and ADO

ADO helps build Web applications that access data in Microsoft® SQL Server[™] 2000. With Microsoft Visual InterDev[™] as a Web development environment, you can use ADO connection and data access routines from within your Microsoft Visual Basic® Scripting Edition or Microsoft JScript® code in your client .htm or server .asp pages. You can encapsulate ADO routines into business objects that perform specific functions, such as validation and authentication.

An example of a Web application that uses ADO to communicate with a SQL Server 2000 database is the Northwind Inventory Management sample. This online inventory management application allows users to view Northwind database inventory-related tables, make product updates, add new products, and remove old ones.

See Also

ADO Web Application

Adding a Data Source

With ADO, you can connect to an instance of Microsoft® SQL Server[™] by using the following types of data sources:

- Microsoft Data Links, using the Microsoft OLE DB Provider for SQL Server (SQLOLEDB).
- ODBC data sources, using the Microsoft OLE DB Provider for ODBC (MSDASQL).

Adding a Microsoft Data Link

You can add a Microsoft Data Link by using Microsoft Windows® Explorer.

To add a Microsoft Data Link by using Windows Explorer

- 1. In Windows Explorer, select the folder in which to add the new data link.
- 2. On the **File** menu, point to **New**, and then click **Text Document**.
- 3. Rename the file in the form Filename.udl.
- 4. Double-click the new file to open the **Data Link Properties** window.
- 5. Select the **Provider** tab, select **Microsoft OLE DB Provider for SQL Server**, and then select the **Connection** tab.
- 6. Specify a server name, the login type, and the default database.

Adding an ODBC Data Source

You can add a data source by using ODBC Administrator, programmatically (by using **SQLConfigDataSource**), or by creating a file.

To add a data source by using ODBC Administrator

- 1. On the **Start** menu, point to **Settings**, and then click **Control Panel**.
- 2. Double-click **ODBC Data Sources (32bit)** (if using Windows 95 or Windows 98) or **Data Sources (ODBC)** (if using Windows NT 4.0 or Windows 2000), click the **User DSN**, **System DSN**, or **File DSN** tab, and then click **Add**.
- 3. Click **SQL Server**, and then click **Finish**.

Complete the steps in the Create a New Data Source to SQL Server Wizard.

Deleting a Data Source

Data sources can be deleted by:

- Deleting the .udl file if the data source is a Microsoft Data Link.
- Using ODBC Administrator if the data source is an ODBC data source.

Deleting a Microsoft Data Link

To delete a Microsoft Data Link file by using Windows Explorer

- 1. In Windows Explorer, select the Microsoft Data Link file.
- 2. Click **Delete**, and then click **Yes** to confirm the deletion.

Deleting an ODBC Data Source

To delete a data source by using ODBC Administrator

- 1. On the **Start** menu, point to **Settings**, and then click **Control Panel**.
- Double-click ODBC Data Sources (32bit) (if using Windows 95 or Windows 98) or Data Sources (ODBC) (if using Windows NT 4.0 or Windows 2000), and then click the User DSN, System DSN, or File DSN tab.
- 3. Click the data source to delete, click **Remove**, and then click **Yes** to confirm the deletion.

Creating an ADO Application

The following components and functions are part of the ADO architecture.

Component	Function
Application	Calls ADO objects, collections, methods, and properties
	to communicate with a data source. Submits SQL
	statements, and processes result sets.
ADO	Manages communication between an application and the
	OLE DB provider used by the application.
OLE DB	Processes all ADO calls from the application, connects
provider	to a data source, passes SQL statements from the
	application to the data source, and returns results to the
	application.
Data source	Contains the information used by a provider to access a
	specific instance of data in a DBMS.

An application that uses ADO to communicate with Microsoft® SQL Server[™] 2000 performs the following tasks:

- Connects with a data source.
- Sends SQL statements to the data source.
- Processes the results of statements from the data source.
- Processes errors and messages.
- Terminates the connection to the data source.

A more complex application written using ADO can also perform the following tasks:

• Use cursors to control location in a result set.

- Execute stored procedures on a server.
- Execute user-defined functions on a server.
- Manage queries that generate multiple result sets.
- Request commit or rollback operations for transaction control.
- Perform catalog operations to inquire about the attributes of a result set.
- Manage long data (**text**, **ntext**, and **image** columns) operations.
- Perform XML operations using XPath queries, annotated schemas, and Transact-SQL extensions such as FOR XML and OpenXML.

For more information, see <u>Using ADO in Different Development Environments</u>.

Connecting to a SQL Server Data Source

ADO can use any OLE DB provider to establish a connection. The provider is specified through the **Provider** property of the **Connection** object. Microsoft® SQL Server[™] 2000 applications use SQLOLEDB to connect to an instance of SQL Server, although existing applications can also use MSDASQL to maintain backward compatibility.

Using the **Execute** method of the **Connection** object is one way to execute an SQL statement against a SQL Server data source.

The **Connection** object allows you to:

- Configure a connection.
- Establish and terminate sessions with data sources.
- Identify an OLE DB provider.
- Execute a query.
- Manage transactions on the open connection.
- Choose a cursor library available to the data provider.

There are some differences in connection properties between SQLOLEDB and MSDASQL. For information about connection properties for MSDASQL, see the MSDN Library at <u>Microsoft Web site</u>.

If you are writing a connection string for use with SQLOLEDB:

- Use the **Initial Catalog** property to specify the database.
- Use the **Data Source** property to specify the server name.

• Use the **Integrated Security** keyword, set to a value of **SSPI**, to specify Windows Authentication (recommended), or use the **User ID** and **Password** connection properties to specify SQL

Server Authentication.

If you are writing a connection string for use with MSDASQL:

- Use the **Database** keyword or **Initial Catalog** property to specify the database.
- Use the **Server** keyword or **Data Source** property to specify the server name.
- Use the Trusted_Connection keyword, set to a value of yes, to specify Windows Authentication (recommended), or
 Use the UID keyword or User ID property, and the Pwd keyword or Password property to specify SQL Server Authentication.

For more information about a complete list of keywords available for use with a SQLOLEDB connection string, see <u>Connection Object</u>.

Restrictions on Multiple Connections

SQLOLEDB does not allow multiple connections. Unlike MSDASQL, SQLOLEDB does not attempt to reconnect when the connection is blocked.

Examples

A. Using SQLOLEDB to connect to an instance of SQL Server: setting individual properties

The following Microsoft Visual Basic® code fragments from the ADO Introductory Visual Basic Sample show how to use SQLOLEDB to connect to an instance of SQL Server. ' Initialize variables. Dim cn As New ADODB.Connection

```
• • •
```

Dim ServerName As String, DatabaseName As String, _ UserName As String, Password As String

' Put text box values into connection variables. ServerName = txtServerName.Text DatabaseName = txtDatabaseName.Text UserName = txtUserName.Text Password = txtPassword.Text

```
' Specify the OLE DB provider.
cn.Provider = "sqloledb"
```

```
' Set SQLOLEDB connection properties.
cn.Properties("Data Source").Value = ServerName
cn.Properties("Initial Catalog").Value = DatabaseName
```

```
' Decision code for login authorization type:
' Windows NT or SQL Server authentication.
If optWinNTAuth.Value = True Then
```

cn.Properties("Integrated Security").Value = "SSPI" Else

```
cn.Properties("User ID").Value = UserName
cn.Properties("Password").Value = Password
End If
```

' Open the database. cn.Open

B. Using SQLOLEDB to connect to an instance of SQL Server: connection string method

The following Visual Basic code fragment shows how to use SQLOLEDB to connect to an instance or SQL Server:

' Initialize variables. Dim cn As New ADODB.Connection Dim provStr As String

```
' Specify the OLE DB provider.
cn.Provider = "sqloledb"
```

' Specify connection string on Open method. ProvStr = "Server=MyServer;Database=northwind;Trusted_Connection cn.Open provStr

C. Using MSDASQL to connect to an instance of SQL Server

To use MSDASQL to connect to an instance of SQL Server, use the following types of connections.

The first type of connection is based on the ODBC API **SQLConnect** function. This type of connection is useful in situations where you do not want to code specific information about the data source. This may be the case if the data source could change or if you do not know its particulars.

In the code fragment shown, the **ConnectionTimeout** method sets the connection time-out value to 100 seconds. Next, the data source name, user ID, and password are passed as parameters to the **Open** method of the **Connection** object, using an ODBC data source named MyDataSource that points to the **northwind** database on an instance of SQL Server. The **sa** login ID is provided as the second parameter and the password is the third parameter.

Dim cn As New ADODB.Connection

cn.ConnectionTimeout = 100
' DSN connection. You can use variables for the parameters.
cn.Open "MyDataSource", "sa", "MyPassword"
' Alternative syntax follows:

'cn.Open "DSN=DataSourceName;UID=sa;PWD=Password;"

cn.Close

The second type of connection is based on the ODBC API **SQLDriverConnect** function. This type of connection is useful in situations where you want a driver-specific connection string. To make a connection, use the **Open** method of the **Connection** object and specify the driver, server name, user ID, password, and database. You can also specify any other valid keywords to include in the connection string. For more information about the keyword list, see <u>SQLDriverConnect</u>.

Dim cn As New ADODB.Connection

' Connection to SQL Server without using ODBC data source. cn.Open "Driver={SQL Server};Server=Server1;Uid=SA;Pwd=;Datab

cn.Close

See Also

ADO Connection and Error Handling

Connecting to Multiple Instances of SQL Server

Multiple instances of Microsoft® SQL Server[™] 2000 can be run on one computer. The computer can support a default instance of SQL Server and additional named instances of SQL Server. An application connects to the default instance of SQL Server by specifying the name of the computer. To connect to a named instance, the application specifies both the computer name and the instance name using this format: '<computername>\<instancename>'

Examples

A. Using ADO and SQLOLEDB to connect to a default instance of SQL Server

The following Microsoft Visual Basic[®] code fragment shows use ADO and SQLOLEDB to connect to a default instance of SQL Server.

'Initialize variables. Dim cn As New ADODB.Connection Dim provStr As String 'Specify the OLE DB provider. cn.Provider = "sqloledb" 'Specify a connection string for the default instance 'of SQL Server. ProvStr = "Server=NorthRegion;Database=northwind;UID=sa;pwd=;" cn.Open ProvStr

B. Using ADO and SQLOLEDB to connect to a named instance of SQL Server

The following Visual Basic code fragment shows how to use ADO and SQLOLEDB to connect to a named instance of SQL Server 2000.

Note To connect to an instance of SQL Server, you must have the latest version of Microsoft Data Access Components (MDAC) installed on both computers.

The latest version of MDAC is installed automatically with SQL Server 2000; however, if you are using SQL Server 7.0, 6.5, or 6.0, you need to install the latest version of MDAC.

'Initialize variables. Dim cn As New ADODB.Connection Dim provStr As String 'Specify the OLE DB provider. cn.Provider = "sqloledb" 'Specify a connection string for an additional instance 'of SQL Server. ProvStr = "Server=NorthRegion\Inst02;Database=northwind;UID=sa;I cn.Open ProvStr

Note To connect to a named instance using JScript, use this format: '<computername>\\<instancename>'

Retrieving Connection Properties

The **Properties** collection and **Property** object provide information about the characteristics of the **Connection**, **Command**, **Recordset**, and **Field** objects. The **Properties** collection can be accessed through any of these objects, and the **Property** object can be accessed through the **Properties** collection by using the default indexing method.

Examples

A. Retrieving the ConnectionTimeout, CommandTimeout, and Updatability properties.

The **Properties** collection is retrieved through the **Connection**, **Command**, and **Recordset** objects. The **ConnectionTimeout** property of the **Connection** object is then printed. The same steps are performed for the **Command** and **Recordset** objects.

This example demonstrates how to retrieve connection properties.

Dim cn As New ADODB.Connection Dim cmd As New ADODB.Command Dim rs As New ADODB.Recordset

```
cn.Provider = "sqloledb"
cn.Properties("Data Source").Value = "MyServerName"
cn.Properties("Initial Catalog").Value = "northwind"
cn.Properties("Integrated Security").Value = "SSPI"
cn.Open
```

'Retrieve the ConnectionTimeout property. Debug.Print cn.Properties("ConnectionTimeout")

Set Cmd.ActiveConnection = Cn cmd.CommandText = "titles" cmd.CommandType = adCmdTable
Set rs = cmd.Execute

'Retrieve the CommandTimeout property. Debug.Print cmd.Properties("CommandTimeout")

'Retrieve the Updatability property. Debug.Print rs.Properties("Updatability")

Executing Queries

After an ADO application connects with a data source, it can execute SQL statements on the data source. The general sequence of events in executing an SQL statement is:

- 1. Construct the statement.
- 2. Execute the statement.
- 3. Retrieve any result sets.

After an application retrieves all of the rows in all of the result sets returned by the SQL statement, it can execute another query using the same connection. If an application does not need to retrieve all of the rows in a particular result set, it can cancel the remainder of the result set by calling the **Close** method to close the **Connection** object. This closes any active **Recordset** objects associated with the connection.

If an ADO application must execute the same SQL statement multiple times with different data, you can use the **Parameters** collection, which consists of **Parameter** objects that provide parameter information and data to the **Command** object.

In addition to executing SQL statements, an application can:

- Execute stored procedures.
- Execute user-defined functions.
- Perform batch updates.
- Generate multiple recordsets.

Using the Command Object

An application can use the **Command** object to issue commands to the database. These commands include query strings, prepared query strings, and associated parameters. The actual command language and features supported depend on the underlying OLE DB provider.

The **Command** object can either open a new connection or use an existing connection to perform queries, depending on what is specified in the **ActiveConnection** property of the **Command** object:

- If the **ActiveConnection** property is set with a reference to a **Connection** object, the **Command** object uses the existing connection.
- If the **ActiveConnection** property is set with a connection string, a new connection is established.

More than one **Command** object can use the connection from the same **Connection** object.

Executing commands can generate zero, one, or multiple recordsets. For example, executing a data definition language query does not generate a recordset. Executing one SELECT statement can generate a recordset, and executing a batch of SELECT statements or a stored procedure can generate more than one recordset.

Execute Method

Use the **Execute** method of the **Command** object to execute a query, data definition command, or stored procedure. The syntax is:

Set rs = cmd.Execute(NumRecords, Parameters, Options)

The variable *rs* is the returned **Recordset** object, and the parameters are optional. The *NumRecords* parameter specifies the number of rows returned; *Parameters* is a variant that specifies initial input parameter values; and *Options* specifies the type of query (in the form of a **CommandTypeEnum** constant), if known, to optimize processing.

Command Type Options

Command type options are specified in the **CommandType** property. A command can be a standard SQL data manipulation language statement, such as SELECT, INSERT, UPDATE, or DELETE, or any data definition language statement, such as CREATE or DROP. A command can also be the name of a stored procedure or table.

CommandTypeEnum	
Constant	Query String
adCmdFile	File name of a persistently stored
	Recordset object
adCmdStoreProc	Stored procedure
adCmdTable	Table name
adCmdTableDirect	Table name whose columns are all returned
adCmdText	SQL statement
adCmdUnknown	Contents of the command are not known
	(default)
adCmdUnspecified	Unspecified command type argument

The **CommandType** property has the following values.

Prepared Property

You can prepare query strings using the **Prepared** property. Setting the **Prepared** property allows a query plan to be created when it is first executed. The query plan is then used for subsequent executions to enhance performance. A query string should be prepared only when executed more than one time because it may take more time to create a query plan than to execute the query string directly. Performance is enhanced only when you execute the query string more than one time.

The **Prepared** property can also be useful when executing a parameterized query string repeatedly. Different parameter values can be substituted each time it is executed instead of reconstructing the query string. The **Parameter** object can be created using the **CreateParameter** method.

See Also

Command Object

Using the Connection Object

In addition to the **Command** object, an application can use the **Connection** object to issue commands, stored procedures, and user-defined functions to a database as if they were native methods on the **Connection** object. To execute a query without using a **Command** object, an application can pass a query string to the **Execute** method of a **Connection** object.

However, a **Command** object is required if you want to save and re-execute the command text, or use query parameters.

To execute a command on the Connection object

- 1. Assign a name to the command using the **Name** property of the **Command** object.
- 2. Set the **ActiveConnection** property of the **Command** object to the connection.
- 3. Issue a statement where the command name is used as if it were a method on the **Connection** object, followed by any parameters.
- 4. Create a **Recordset** object if any rows are returned.
- 5. Set the **Recordset** properties to customize the resulting **Recordset**.

Using the Connection Object to Execute Commands

This example shows how to use the **Execute** method of the **Connection** object to execute commands.

Dim cn As New ADODB.Connection

Dim rs As New ADODB.Recordset

cmd1 = txtQuery.Text
Set rs = cn.Execute(cmd1)

After the **Connection** and **Recordset** objects are created, the variable *cmd1* is assigned the value of a user-supplied query string (txtQuery.Text) from a Microsoft Visual Basic® form. The recordset is assigned the results of a query, by calling the **Execute** method of the **Connection** object, with the variable *cmd1* used as the query string parameter.

See Also

Connection Object

Constructing an SQL Statement

ADO applications perform much of their database access by executing SQL statements. The form of these statements depends on the needs of the application. SQL statements can be constructed in the following ways:

- Hard-coded
- Constructed at run time

Hard-coded SQL statements are static statements performed by an application as a fixed task.

SQL statements constructed at run time enable the user to tailor the statement by using common clauses, such as SELECT, WHERE, and ORDER BY. This includes ad hoc queries entered by users.

The column list in a SELECT statement should contain only the columns needed to perform the current task. This reduces the amount of data sent over the network, and it reduces the effect of database changes on the application. For example, if an application does not reference a column from a table, the application is not affected by any changes made to that column.

Constructing SQL Statements for Cursors

The set of rows returned by a SELECT statement consists of all the rows that satisfy the conditions in the WHERE clause of the statement, and is known as the result set. Because ADO applications cannot always work effectively with the entire result set as a unit, they must use either ADO client-side cursors or SQL Server server-side cursors to work with a smaller subset of rows. For more information, see <u>Cursors</u> and <u>Using Cursors with ADO</u>.

Using Parameters

Prepared statements, stored procedures, and user-defined functions may require the use of parameters. The **Parameters** collection, which consists of **Parameter** objects, provides parameter information and data for the **Command** object. You use the **Parameters** collection and **Parameter** objects when the query in the **Command** object requires parameters.

A **Parameter** object can serve as an input parameter, an output parameter data, or a return value. The **Refresh** method of the **Parameters** collection can force providers to update parameter information; however, this operation can take some time to complete.

The **Parameters** collection provides parameter information and data for the **Command** object. You use the **Parameters** collection and **Parameter** objects when the query in the **Command** object requires parameters.

This example shows the creation of an input parameter for a stored procedure using Transact-SQL syntax:

```
USE NORTHWIND
GO
drop proc myADOParaProc
GO
CREATE PROC myADOParaProc
@categoryid int(4)
AS
SELECT * FROM products WHERE categoryid = @categoryid
GO
```

The **myADOParaProc** stored procedure performs a SELECT query against the **products** table of the **northwind** database, taking one **@categoryid** input parameter in its WHERE clause. The data type for the **@category** parameter is **int**, and its size is 4.

Here is the Microsoft® Visual Basic® code:

```
Dim cn As New ADODB.Connection
Dim cmd As New ADODB.Command
Dim rs As New ADODB.Recordset
Dim prm As ADODB.Parameter
Dim fld As ADODB.Field
Dim provStr As String
```

```
' Connect using the SQLOLEDB provider.
cn.Provider = "sqloledb"
```

' Specify connection string on Open method. provStr = "Server=MyServer;Database=northwind;Trusted_Connection cn.Open provStr

```
' Set up a command object for the stored procedure.
Set cmd.ActiveConnection = cn
cmd.CommandText = "myADOParaProc"
cmd.CommandType = adCmdStoredProc
cmd.CommandTimeout = 15
```

' Set up a new parameter for the stored procedure. Set prm = Cmd.CreateParameter("CategoryID", adInteger, adParamInp Cmd.Parameters.Append prm

```
' Create a recordset by executing the command.
Set rs = cmd.Execute
Set Flds = rs.Fields
```

```
' Print the values for all rows in the result set.
While (Not rs.EOF)
For Each fld in Flds
Debug.Print fld.Value
Next
```

```
Debug.Print ''''
rs.MoveNext
Wend
```

' Close recordset and connection. rs.Close cn.Close

The **myADOParaProc** stored procedure expects an input parameter with a data type of **int** and a size of 4. The **CreateParameter** method is used to create a **Parameter** object with the following characteristics: the data type is **adInteger** for an integer, the parameter type is **adParamInput** for input parameter, and the data length is 4. This **Parameter** object is also given the name CategoryID. The data value 7 (one of the possible values of CategoryID in the **products** table) is hard-coded.

After the parameter is specified, the **Append** method adds the **Parameter** object to the **Parameters** collection. The **myADOParaProc** stored procedure is executed, and a **Recordset** object is created. The values for the columns of each row in the recordset are printed, and the **Connection** and **Recordset** objects are closed.

See Also

Using Return Code and Output Parameters for Stored Procedures

Executing Statements

An ADO application can execute an SQL statement in the following ways:

- Direct execution
- Prepared execution

These methods of execution can be used for one SQL statement, a call of a stored procedure or user-defined function, or a batch of SQL statements.

Executing Statements Directly

Direct execution is the most basic way to execute a statement and is commonly used by applications that build and execute statements at run time. It is the most efficient method for using statements that will be executed a single time or for calling stored procedures. One drawback of direct execution is that a SQL statement must be parsed and compiled every time it is executed, which increases overhead if the statement is executed a number of times.

An application builds a character string containing an SQL statement and submits it for execution using the **Execute** method of the **Command** or **Connection** object. When the statement reaches the server, Microsoft® SQL ServerTM 2000 compiles it into an execution plan and then immediately runs the execution plan.

For SQL Server 2000 applications, using the **Execute** method with parameter markers for commonly executed SQL Statements can approach the efficiency of prepared execution.

Executing Prepared Statements

Prepared execution is commonly used by applications to execute the same parameterized SQL statement repeatedly. Prepared execution is faster than direct execution for statements executed more than three or four times because the statement is compiled only once, while statements executed directly are compiled each time they are executed. Prepared execution can also provide a reduction in network traffic because the driver can send an execution plan identifier and the parameter values, rather than an entire SQL statement, to the data source each time the statement is executed. The **Prepared** property of the **Command** object allows you to specify whether to prepare a statement.

An ADO application can use prepared execution to reduce the parsing and compiling overhead associated with repeatedly executing an SQL statement that is executed numerous times. The application builds a character string containing an SQL statement and then uses the **Prepared** property to have the provider save a prepared (or compiled) version of the query specified in the **CommandText** property before the first execution of a **Command** object. This can slow the first call of the Execute method, but after the command is compiled, the provider uses the compiled version of the command for any subsequent executions, which results in improved performance.

If the Prepared property is set to False, the provider executes the **Command** object directly without creating a compiled version.

The **Prepared** property can be used when executing a statement with multiple parameter sets. An application can execute a parameterized statement more than once by supplying a different parameter set at each execution instead of reconstructing the statement whenever the parameter set is different.

Microsoft® SQL Server[™] 2000 continues to support the prepare/execute model of OLE DB and ODBC. For applications using the Microsoft OLE DB Provider for ODBC (MSDASQL), this option can be disabled through the **SQL Server ODBC Data Source Setup** dialog box if an ODBC data source is used to connect to an instance of SQL Server. If the option is disabled, the SQL statement is stored and then sent to the server each time it is executed.

This example shows using a prepared statement to update a query and construct the query dynamically with a different set of parameters at execution time.

```
Dim cn As New ADODB.Connection
Dim cmdPrep1 As New ADODB.Command
Dim prm1 As New ADODB.Parameter
Dim prm2 As New ADODB.Parameter
Dim strCn As String
```

```
strCn = "Server=MyServerName;Database=pubs;Trusted_Connection=
cn.Provider = "sqloledb"
cn.Open strCn
Set cmdPrep1.ActiveConnection = cn
cmdPrep1.CommandText = "UPDATE titles SET type=? WHERE title
cmdPrep1.CommandType = adCmdText
cmdPrep1.Prepared = True
```

Set prm1 = cmdPrep1.CreateParameter("Type", adChar, adParamInput cmdPrep1.Parameters.Append prm1

Set prm2 = cmdPrep1.CreateParameter("ProductID", adInteger, adPara cmdPrep1.Parameters.Append prm2

cmdPrep1.Execute

cmdPrep1("Type") = "New Cook"
cmdPrep1("title_id") = "TC7777"
cmdPrep1.Execute

cn.Close

Data is updated in the **titles** table by using different parameter values. The query string is prepared so that different sets of parameters can be supplied. Two parameters are required for the update operation: *type* and *title_id*. They are

created by the two **CreateParameter** methods and appended to the **Parameters** collection with the **Append** method.

The first set of parameters has the values New Bus and BU7832. Because the **Prepared** property is set to TRUE, different values can be supplied to *cmdPrep1* without reconstructing and re-executing the query string.

Note Prepared statements cannot be used to create temporary objects on SQL Server. Prepared statements cannot reference system stored procedures that create temporary objects, such as temporary tables. An application must directly execute these procedures.

Executing Stored Procedures

A stored procedure is a precompiled executable object that contains one or more SQL statements. Stored procedures can have input and output parameters and can issue an integer return code.

Executing a stored procedure is similar to executing a prepared statement, except that the stored procedure exists as a permanently compiled object in the database. A stored procedure can also be used to hide complex SQL statements from the application.

When executing a stored procedure in a **Command** object, the **CommandType** property must be specified with the **adCmdStoredProc** value. With **adCmdStoredProc**, the corresponding SQL statement for the underlining provider is generated. For applications that use the Microsoft OLE DB Provider for ODBC (MSDASQL), ODBC escape sequences for procedure calls are generated.

There is no need to prepare a statement that calls only a stored procedure. Both stored procedures and prepared statements are methods of precompiling statements. Because a stored procedure is precompiled, preparing a stored procedure call adds overhead. The prepared statement adds a small precompiled execution plan that calls the stored procedure execution plan, rather than executing the stored procedure execution plan directly.

This example shows the execution of the **sp_who** SQL Server system stored procedure:

Dim cn As New ADODB.Connection Dim cmd As New ADODB.Command Dim rs As New ADODB.Recordset

cn.Provider = "sqloledb" cn.Properties("Data Source").Value = "MyServerName" cn.Properties("Initial Catalog").Value = "northwind" cn.Properties("Integrated Security").Value = "SSPI" cn.Open Cmd.ActiveConnection = cn Cmd.CommandText = "sp_who" Cmd.CommandType = adCmdStoredProc

Set rs = Cmd.Execute Debug.Print rs(0) rs.Close

See Also

Calling a Stored Procedure (OLE DB)

Using Return Code and Output Parameters for Stored Procedures

Stored procedures can contain input parameters, output parameters, and return values. You specify input parameters, output parameters, and return values for a stored procedure through the **Parameter** object. In the case of output parameters and return values, the values are not returned until the data of the **Recordset** object has been fetched completely or the **Recordset** has been closed.

The following stored procedure contains one input parameter, one output parameter, and a return parameter. The procedure selects those rows in the **titles** table of the **pubs** database where the royalty percent paid to the author is greater than the amount entered by the user (the input parameter). The program returns the number of rows as the output variable. If the program returns any rows, a return code of 0 is issued; if no rows are returned, a return code of 99 is issued.

```
USE pubs
GO
CREATE PROCEDURE myProc
                OUTPUT
@outparm
           int
@inparm
          int
AS
SELECT * FROM titles WHERE royalty > @inparm
SELECT @outparm = COUNT (*) FROM TITLES WHERE royalty >
IF (@outparm > 0)
RETURN 0
ELSE
RETURN 99
GO
```

An ADO code program that executes the stored procedure **myProc** is shown here.

Dim cn As New ADODB.Connection

Dim cmd As New ADODB.Command Dim rs As New ADODB.Recordset Dim fldloop As ADODB.Field Dim param1 As Parameter, param2 As Parameter, param3 As Parameter Dim provStr As String Dim royalty As Variant

Private Sub spStart()

' Connect using the SQLOLEDB provider. cn.Provider = "sqloledb"

' Specify connection string on Open method.
provStr = "Server=MyServer;Database=pubs;Trusted_Connection=yes
cn.Open provStr

' Set up a command object for the stored procedure. Set cmd.ActiveConnection = cn cmd.CommandText = "myProc" cmd.CommandType = adCmdStoredProc

'Set up a return parameter.
Set param1 = cmd.CreateParameter("Return", adInteger, adParamRetur
cmd.Parameters.Append param1

' Set up an output parameter. Set param2 = cmd.CreateParameter("Output", adInteger, adParamOutp cmd.Parameters.Append param2

```
' Set up an input parameter.
Set param3 = cmd.CreateParameter("Input", adInteger, adParamInput)
cmd.Parameters.Append param3
royalty = Trim(InputBox("Enter royalty:"))
```

param3.Value = royalty

' Execute command, and loop through recordset, printing out rows. Set rs = cmd.Execute

```
Dim i As Integer
While Not rs.EOF
For Each fldloop In rs.Fields
Debug.Print rs.Fields(i)
i = i + 1
Next fldloop
Debug.Print ""
i = 0
rs.MoveNext
Wend
```

```
' Need to close recordset before getting return
' and output parameters.
rs.Close
```

Debug.Print "Program ended with return code: " & Cmd(0) Debug.Print "Total rows satisfying condition: " & Cmd(1) cn.Close

End Sub

The following parameters are needed for the **myProc** stored procedure:

A return parameter to hold the return value (0 or 99). The return parameter is created as a return type of parameter adParamReturnValue, and the data type is adInteger for integer. Because the return parameter is the first parameter added to the collection, its index value is zero, and it can be dereferenced through that index (for example, as Cmd(0)).

- An output parameter to hold the value of the count of the number of returned rows. The output parameter is created as **adParamOuput** for the output parameter type, and the data type is **adInteger** for **integer**. Because the output parameter is the second parameter added to the collection, its index value is 1, and it can be dereferenced through that index (for example, as **Cmd(1)**).
- An input parameter, which holds the value of the user-supplied percent royalty number. The input parameter is created as **adParamInput** for the input parameter type, and the data type is **adInteger** for **integer**.

Because the data type of these stored procedure parameters is **integer**, there is no need to specify the data length as a parameter when defining them with the **CreateParameter** method.

After each parameter is added to the **Parameters** collection, executing the query string creates a recordset. After the recordset is closed, the values for the return code and output parameters are available.

Executing User-Defined Functions

Executing a user-defined function is similar to executing a prepared Transact-SQL statement, except that the user-defined function exists as a permanent object in the database. Executing a user-defined function can increase the efficiency of an application because it can reference complex Transact-SQL statements at the server instead of from an application.

This example shows the execution of the **fn_helpcollations** built-in, userdefined function. All user-defined functions can be executed using the technique demonstrated in this example.

Dim cn As New ADODB.Connection Dim cmd As New ADODB.Command Dim rs As New ADODB.Recordset

```
cn.Open "Provider=sqloledb;Data Source=MyServerName;" & _
"Initial Catalog=northwind;User Id=sa;Password=;"
```

```
'Prepare the user-defined function statement and execute the command
Cmd.ActiveConnection = cn
Cmd.CommandText = "select * from ::fn_helpcollations()"
Set rs = Cmd.Execute
```

rs.Close

Using Batch Updates

The **Update** method of the **Recordset** object allows you to update the current row. The **UpdateBatch** method applies all pending new, updated, and deleted rows to the **Recordset** object. Using a **LockType** property value of **adLockBatchOptimistic**, the **UpdateBatch** method allows you to commit all pending changes at the client and send all the changes to the database at one time. The pending changes can be canceled by calling the **CancelBatch** method.

With the **UpdateBatch** method, an error is returned if all the changes fail to be applied to the database. If only some of the changes fail, a warning is returned instead of an error, by using the **Errors** collection and **Error** object.

The **UpdateBatch** method is valid only when the **LockType** property is specified with **adLockBatchOptimistic** and the cursor type is either keyset-driven or static. The keyset-driven cursor can be supported only with tables that have unique indexes.

This example shows the use of the **UpdateBatch** method to apply all pending changes; it creates a recordset by using the keyset-driven cursor with the **LockType** property set to **adLockBatchOptimistic**. After the **Recordset** object is created, the user is prompted to change any row in the **titles** table of **pubs** with a type of psychology to self help. Clicking **OK** commits the changes using the **UpdateBatch** method; clicking **No** cancels the changes using the **CancelBatch** method. The routine at the end restores the original values to the table.:

Public Sub UpdateBatchX()

Dim rstTitles As ADODB.Recordset Dim strCnn As String Dim strTitle As String Dim strMessage As String

' Assign connection string to variable.
strCnn = "Provider=sqloledb;" & _
"Data Source=srv;Initial Catalog=pubs;User Id=sa;Password=; "

```
Set rstTitles = New ADODB.Recordset
rstTitles.CursorType = adOpenKeyset
rstTitles.LockType = adLockBatchOptimistic
rstTitles.Open "titles", strCnn, , , adCmdTable
```

rstTitles.MoveFirst

```
' Loop through recordset, and prompt user for
' change of type for a specified title.
Do Until rstTitles.EOF
If Trim(rstTitles!Type) = "psychology" Then
strTitle = rstTitles!Title
strMessage = "Title: " & strTitle & vbCr & _
"Change type to self help?"
If MsgBox(strMessage, vbYesNo) = vbYes Then
rstTitles!Type = "self_help"
End If
End If
```

```
rstTitles.MoveNext
Loop
```

```
' Ask if the user wants to commit to all the
' changes made earlier.
If MsgBox("Save all changes?", vbYesNo) = vbYes Then
  rstTitles.UpdateBatch
Else
  rstTitles.CancelBatch
End If
```

' Print current data in recordset.

```
rstTitles.Requery
rstTitles.MoveFirst
Do While Not rstTitles.EOF
Debug.Print rstTitles!Title & " - " & rstTitles!Type
rstTitles.MoveNext
Loop
```

```
'Restore original values because this is a demonstration.
rstTitles.MoveFirst
Do Until rstTitles.EOF
If Trim(rstTitles!Type) = "self_help" Then
rstTitles!Type = "psychology"
End If
rstTitles.MoveNext
Loop
rstTitles.UpdateBatch
```

rstTitles.Close

End Sub

Generating Multiple Recordsets

Microsoft® SQL Server[™] 2000 allows a batch of queries to be issued and executed. When a batch of queries is executed, more than one recordset can be generated. Multiple recordsets can also be generated by SQL statements that include multiple SELECT statements or COMPUTE BY and COMPUTE clauses, or by stored procedures that contain more than one SELECT statement.

Note If you are using a SQL Server API server cursor, you cannot execute a Transact-SQL statement or stored procedure that generates more than one result set. If you need to generate multiple result sets, use a client cursor by leaving the cursor properties of the **Recordset** object set to their defaults (for example, forward only/read-only (**adOpenForwardOnly**) and an editing lock of **adLockReadOnly**).

When multiple recordsets are generated, you need to fetch one recordset at a time until no more recordsets are available. The **NextRecordset** method of the **Recordset** object allows you to fetch subsequent recordsets. If no more recordsets are available, the returned **Recordset** object is set to **Nothing**. Generally, you write code to test whether a **Recordset** object is set to **Nothing** as the test condition for exiting the multiple recordset loop.

The following example shows how to fetch multiple recordsets from a stored procedure using the **NextRecordset** method of the **Recordset** object.

The stored procedure syntax is:

```
DROP PROC myNextproc
GO
CREATE PROC myNextproc AS
SELECT * FROM titles
SELECT * FROM publishers
GO
```

The stored procedure generates two result sets: one for the result of SELECT * FROM **titles** and the other for the result of SELECT * FROM **publishers**.

The ADO code syntax is:

Dim cmd As New ADODB.Command Dim rs As ADODB.Recordset

```
cn.Provider = "sqloledb"
cn.Properties("Data Source") = "MyServerName"
cn.Properties("Initial Catalog") = "pubs"
cn.Properties("user ID") = "sa"
cn.Properties("password") = ""
cn.Open
```

Cmd.CommandText = "myNextProc" Cmd.CommandType = adCmdStoredProc

```
Set rs = Cmd.Execute
While Not rs Is Nothing
If (Not rs.EOF) Then
Debug.Print rs(0)
End If
Set rs = rs.NextRecordset
Wend
```

After the **myNextProc** stored procedure is executed, a **Recordset** object is created. Because two result sets are generated by the **myNextProc** stored procedure, each **Recordset** object can be retrieved by using the **NextRecordset** method. The **Recordset** object, *rs*, is reused for each recordset.

Processing Results

After an application submits an SQL statement, Microsoft® SQL Server[™] 2000 returns any resulting data as one or more result sets. A result set is a set of rows and columns that match the criteria of the query. SELECT statements, catalog functions, and some procedures produce a result set made available to an application in tabular form. If the executed SQL statement is a stored procedure, a batch containing multiple commands, or a SELECT statement containing keywords, such as COMPUTE or COMPUTE BY, there will be multiple result sets to process.

The ADOX **Catalog** object can also retrieve data. For example, The **Catalog** object allows you to manipulate and retrieve data about tables, views, and stored procedures in a SQL Server 2000 database. These result sets can contain zero or more rows. Other SQL statements, such as GRANT or REVOKE, do not return result sets.

Each INSERT, UPDATE, and DELETE statement returns a result set containing only the number of rows affected by the modification. These counts can be canceled by including a SET NOCOUNT ON statement in the batch or stored procedure.

Transact-SQL includes the SET NOCOUNT statement. When the NOCOUNT option is set to ON, SQL Server does not return the counts of the rows affected by a statement.

Several other Transact-SQL statements return their data in messages rather than result sets, such as:

- DBCC
- SET SHOWPLAN
- SET STATISTICS
- PRINT

• RAISERROR

ADO applications use the **Recordset** object to manipulate result sets, and the **Fields** collection and **Field** object to access data in a row. In addition, you can use the **Properties** collection and **Property** object to provide information about the characteristics of a result set.

Using the Recordset Object

The **Recordset** object provides methods for manipulating result sets. It allows you to add, update, delete, and scroll through rows in the recordset.

A **Recordset** object can be created using the **Execute** method of the **Connection** or **Command** object.

Each row in a recordset can also be retrieved and updated using the **Fields** collection and the **Field** object. Updates on the **Recordset** object can be in an immediate or batch mode. When a **Recordset** object is created, a cursor is opened automatically.

The **Recordset** object allows you to specify the cursor type and location for fetching the result set. With the **CursorType** property, you can specify whether the cursor is read-only, forward-only, static, keyset-driven, or dynamic. Cursor type determines if a **Recordset** object can be scrolled or updated and affects the visibility of changed rows. By default, the cursor type is read-only and forward-only.

An application can specify the location of the cursor with the **CursorLocation** property. This property allows you to specify whether to use a client or server cursor. The **CursorLocation** property setting is important when you use disconnected recordsets.

The first part of the **cmdExecute_Click** method in the ADO Introductory Visual Basic Sample shows an example of creating, opening, passing a command string variable to, and positioning the cursor in a recordset.

```
Dim cn As New ADODB.Connection
Dim rs As ADODB.Recordset
...
cmd1 = txtQuery.Text
Set rs = New ADODB.Recordset
rs.Open cmd1, cn
rs.MoveFirst
```

```
•••
```

' Code to loop through result set(s)

See Also

Using Cursors with ADO

Using the Fields Collection and Field Object

The **Fields** collection and **Field** object allow you to access each data column of the current row. The **Fields** collection can be accessed through the **Recordset** object and the **Field** object can be accessed through the **Fields** collection by using the default indexing method. You can use the **Field** object to create a new row or change existing data, and use the **AddNew**, **Update**, or **UpdateBatch** method of the **Recordset** object to apply the new or changed data. An explicit **Edit** method does not need to specified.

This code fragment shows how to use the **Field** object to retrieve the name, type, and values for each data column of the current row. This code assumes you have made a connection and passed an SQL command string to the *cmdText* variable. After the **Recordset** object is created, the **Fields** collection can be retrieved. The example loops through the **Fields** collection to retrieve each **Field** object. The **Name**, **Type**, and **Value** property of each **Field** object is printed.

Dim rs As New ADODB.Recordset Dim fld As ADODB.Field Dim cn As ADODB.Connection Dim cmdText As String

```
cn.Provider = "sqloledb"
cn.Properties("Data Source").Value = "MyServerName"
cn.Properties("Initial Catalog").Value = "northwind"
cn.Properties("Integrated Security").Value = "SSPI"
cn.Open
```

cmdText = "select * from authors"

rs.Open cmdText, cn Set Flds = rs.Fields Dim TotalCount As Integer TotalCount = Flds.Count For Each fld In Flds Debug.Print fld.Name Debug.Print fld.Type Debug.Print fld.Value Next rs.Close

Determining the Characteristics of a Result Set

The **Properties** collection and **Property** object provide information about the characteristics of the **Connection**, **Command**, **Recordset**, and **Field** objects. The **Properties** collection can be accessed through any of these objects, and the **Property** object can be accessed through the **Properties** collection by using the default indexing method.

The **Properties** collection consists of **Property** objects. In addition to returning the value and type for a property, the **Property** object provides attributes of a property. Attributes describe things such as whether the specific property of an object is supported or required, or whether it is read/write or read-only. For example, **ConnectionTimeout** is a property that provides information about the number of seconds to wait to establish a connection before returning a time-out error.

Examples

Enumerating Through the Properties Collection for an Object.

The following code shows a method for listing each property of an object, using a **Connection** object and **Recordset** object as examples.

Dim cn As New ADODB.Connection Dim rs As ADODB.Recordset

cn.Provider = "sqloledb" cn.Properties("Data Source").Value = "MyServerName" cn.Properties("Initial Catalog").Value = "northwind" cn.Properties("Integrated Security").Value = "SSPI" cn.Open

```
Set rs = New ADODB.Recordset
rs.Open "select * from products", cn
```

' Create a variable to list the properties. Dim prop As ADODB.Property

' Enumerate through the properties of the Connection object. For Each prop In cn.Properties

Debug.Print prop.Name, prop.Value, prop.Attributes Next

' Enumerate through the properties of the Recordset object. For Each prop In rs.Properties

Debug.Print prop.Name, prop.Value, prop.Attributes Next

Mapping Data Types

In rowsets and as parameter values, ADO represents data in Microsoft® SQL Server[™] 2000 by using the following data types. The ADO enumerated constant, DataTypeEnum, specifies the data type of the **Field** and **Parameter** objects.

SQL Server Data Type	ADO Data Type
bigint	adBigInt
binary	adBinary
bit	adBoolean
char	adChar
datetime	adDBTimeStamp
decimal	adNumeric
float	adDouble
image	adVarbinary
int	adInteger
money	adCurrency
nchar	adWChar
ntext	adWChar
numeric	adNumeric
nvarchar	adWChar
real	adSingle
smalldatetime	adTimeStamp
smallint	adSmallInt
smallmoney	adCurrency
sql_variant	adVariant
sysname	adWChar
text	adChar
timestamp	adBinary
tinyint	adVarbinary
uniqueidentifier	adGUID
varbinary	adVarbinary

varchar

adChar

ADO supports consumer-requested data conversions as shown in this illustration.

To data type:																													
From data type:	adEmpty	adSmallInt	adinteger	adSingle	adD ouble	adCurrency	adDate	adBSTR	adlDispatch	adError	adBoolean	adVariant	adiUnknown	adD ecimal	adTinyInt	adUnsignedTinyInt	adUnsignedSmall	adUnsignedInt	adBigInt	adUnsignedBigInt	adGUID	adBinary	adChar	adWChar	adNumeric	adDBDate	adDBTime	adDBTimeStamp	adVariant
adEmpty	•	0	0	0	Ο	0	Ο	0			0	0		0	0	0	Ο	0	Ο	0	0		Ο	0	0				
adSmallint	0	•	0	0	0	0	Ο	Ο			0	0		0	0	0	0	Ο	0	0			0	0	0				0
adinteger	Ο	0	•	0	0	0	Ο	Ο			0	0		0	0	0	0	Ο	0	Ο			0	0	0				0
adSingle	0	0	0	•	0	0	Ο	0			0	0		Ο	0	0	0	0	Ο	0			0	0	0				Ο
adDouble	0	0	0	0	•	0	Ο	Ο			0	0		1.000.000	1.			0					0	0	Ο				Ο
adCurrency	0	0	0	0	0	•	0	0			0	0		0	0	0	0	0	0	0			0	0	0				Ο
adDate	Ο	0	0	Ο	Ο	Ο	•	Ο			Ο	0		0	0	0	0	Ο	Ο	Ο			0	Ο	Ο	0	0	0	0
adBSTR	0	0	0	Ο	0	Ο	Ο	•			Ο	Ο		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Ο
adlDispatch	0								•			Ο	0																
adError										•		Ο																	
adBoolean	0	0	0	0	Ο	Ο	Ο	Ο			•	0		0	0	0	0	0	0	0			0	0	0				Ο
adVariant	0	0	0	0	Ο	0	Ο	Ο	0	0	0	•	0	0	Ο	0	Ο	Ο	Ο	Ο	0	0	0	0	0	0	0	0	Ο
adiUnknown	0								Ο			Ο	•																
adDecimal	Ο	0	0	Ο	Ο	Ο	Ο	Ο			Ο	Ο		•	Ο	Ο	Ο	Ο	Ο	Ο			0	Ο	Ο				Ο
adTinyInt	0	0	0	0	0	0	Ο	Ο			0	0		0	•	0	0	Ο	Ο	Ο			0	0	0				Ο
adUnsigned TinyInt	0	0	0	Ο	0	Ο	0	Ο			Ο	0		0	0	•	0	0	0	0			0	0	0				Ο
adUnsigned Small	Ο	0	0	Ο	Ο	0	Ο	Ο			0	0		0	Ο	0	•	0	Ο	0			0	0	0				Ο
adUnsignedInt	0	0	0	0	0	0	Ο	0			0	Ο						•					0	Ο	0				Ο
adBigInt	0																		•	Ο									Ο
adUnsignedBigInt	0																		0	•									
adGUID	Ο							Ο													•		0	Ο					Ο
adBinary	0							0				0						0				•	0	0					Ο
adChar	0	0	0	0	0	0	0	0			0	0		0	0	0	0	0	0	0	0	0	•	0	0				0
adWChar			0	10000	1.000	1.00		1000			0	0		_		-		0											0
adNumeric	0	0	0	0	0	0	0	0			0	0		0	0	0	0	0	0	0			0	0	•				0
adDBDate	0					0	Ο					0											0	0		•	0	Ο	0
adDBTime	0					0	Ο					0											0	0		0	•	Ο	0
adDBTimeStamp	0						0					Ο											0	0		0	0	•	Ο
adVariant		0	0	0	0	0	0	0			0	0		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	•

Supported conversion. No data loss occurs.

O Supported conversion. Data loss can occur.

Data Type Usage Considerations

Microsoft[®] SQL Server[™] 2000 includes the following data types that cannot be used with SQL Server version 7.0 or earlier:

- bigint
- sql_variant

Using the bigint Data Type

The **bigint** data type is an integer containing values from -2^63 (-9,223,372,036,854,775,807) through 2^63-1 (9,223,372,036,854,775,807). The storage size is 8 bytes.

The ADO enumerated constant, DataTypeEnum, specifies the data type of an ADO field, parameter, or property. The DataTypeEnum value, **adBigInt**, has a value of 20, and indicates an 8-byte signed integer, which maps to the SQL Server 2000 **bigint** data type and the OLE DB **DBTYPE_I8** data type.

Using the sql_variant Data Type

The **sql_variant** data type can contain data of any of the SQL Server 2000 data types except those for large objects (**text**, **ntext**, and **image** data types), and the **timestamp** data type. For example, a **sql_variant** column can contain **smallint** values for some rows, **float** values for other rows, and **char/nchar** values in the remainder.

Although there are some restrictions, the **sql_variant** data type is similar to the **variant** data type in Microsoft Visual Basic® and **DBTYPE_VARIANT** in OLE DB. The ADO DataTypeEnum value, **adVariant**, has a value of 12, and maps to the OLE DB **DBTYPE_VARIANT** data type. However, ADO does not yet support this data type completely, and usage may cause unpredictable results.

For more information about support of the **sql_variant** data type by the Microsoft OLE DB Provider for SQL Server (SQLOLEDB), see <u>Data Type</u> <u>Mapping in Rowsets and Parameters</u>.

Using Cursors with ADO

ADO uses both client and server cursors to implement the cursor functionality required by an application. An ADO application controls the cursor behavior by using the **CursorType**, **CursorLocation**, **LockType**, and **CacheSize** properties of the **Recordset** object.

When these properties are set to their default values at the time an SQL statement is executed, the Microsoft OLE DB Provider for SQL Server (SQLOLEDB) does not use a server cursor to implement the result set; instead, it uses a default result set. If any of the values of these properties are changed from their default values at the time an SQL statement is executed, SQLOLEDB attempts to use a server cursor to implement the result set.

Cursor Options with SQL Server

Because ADO allows the setting of cursor properties, the following options exist for using cursors with ADO and Microsoft® SQL Server[™] 2000:

• Leave all cursor properties set to their defaults.

If you use these settings, the provider uses default result set processing (forward only and read-only cursor). The default settings allow a program to execute any Transact-SQL statement; however, only one statement can be active on any connection at a time. The program must either fetch all the rows or cancel the result set before another statement can be executed on the same connection. Following those rules, a program can process Transact-SQL statements or stored procedures that allow multiple result sets.

• Change the default cursor type or lock type.

The provider uses SQL Server API server cursors to deliver the requested cursor functionality. Although this option provides a wide range of cursor functionality, it introduces some restrictions. For example, you cannot execute any Transact-SQL statement, batch, or stored procedure that returns more than one result set. However, it is possible to have multiple active statements on one connection (there can be pending results in the statement handle), provided they are all executed with API server cursors.

• Use an ADO client cursor (set the **CursorLocation** property to **adUseClient**).

ADO implements the cursor; therefore, the application can use only the capabilities supported by the ADO client cursors. The application cannot access the cursor capabilities of the underlying provider. Only a **CursorType** property of **adOpenStatic** (static cursor) is supported for a setting of **adUseClient**.

ADO Cursor Settings

An ADO application can control the cursor functionality using these **Recordset** properties.

Property	Description							
CursorType	Default: adOpenForwardOnly							
	Indicates the type of cursor used:							
	Forward-only/read-only (adOpenForwardOnly)							
	Static (adOpenStatic)							
	Keyset (adOpenKeyset)							
	Dynamic (adOpenDynamic)							
CursorLocation	Default: adUseServer							
	Sets or returns the location of the cursor engine. If you							
	set this property to adUseClient , you can open only a							
	static cursor.							
LockType	Default: adLockReadOnly							
	Indicates the type of locks placed on rows during							
	editing.							
CacheSize	Default: 1							
	Controls how many rows the provider keeps in its							
	buffer and how many rows to retrieve at one time into							
	local memory.							

See Also

<u>Cursors</u>

Using Default Result Sets

By default, an ADO application does not use Microsoft® SQL Server[™] 2000 API server cursors with SQLOLEDB. The default cursor used by the ADO application is read-only and forward-only, and uses default result set processing.

Default result sets support all of the Transact-SQL statements. There are no restrictions on the types of SQL statements that can be executed when using a default result set. However, server cursors do not support all Transact-SQL statements. For example, server cursors do not support any SQL statement that generates multiple result sets.

The following types of statements are not supported by server cursors:

• Batches. These are SQL statements built from two or more individual SQL SELECT statements. For example:

SELECT * FROM authors; SELECT * FROM titles

- Stored procedures with multiple SELECT statements. These are SQL statements that execute a stored procedure containing more than one SELECT statement. This includes SELECT statements that fill parameters or variables.
- Keywords These are SQL statements containing the keywords COMPUTE, COMPUTE BY, FOR BROWSE, or INTO.

In SQL Server 2000, if an SQL statement that matches any of these types is executed with a server cursor, the server cursor is implicitly converted to a default result set. An application can call the **Supports** method of the **Recordset** object to verify the specific functionality of the cursor setting. For more information, see <u>Implicit Cursor Conversions</u>.

SQL statements that do not fit the types listed earlier can be executed with any statement settings; they work equally well with either a default result set or a server cursor.

Using Server Cursors with ADO

ADO and OLE DB map cursors over the result sets of executed SQL statements. SQLOLEDB implements these operations using server cursors, which are cursors implemented on the server and managed by API cursor functions.

Server Cursor Details

To use a server cursor, an application can set these properties to anything other than the default value:

- Set the cursor type of the **Recordset** object to **adOpenKeyset**, **adOpenDynamic**, or **adOpenStatic**.
- Set the **LockType** of the **Recordset** object to **adLockPessimistic**, **adLockOptimistic**, or **adLockBatchOptimistic**.
- Set the **CacheSize** property to anything other than the default value of 1.

The **CursorLocation** property should remain at the default setting, **adUseServer**.

Server cursors are created only for statements that begin with:

- SELECT
- EXEC[ute] *procedure_name*
- **call** procedure_name

Even if an application explicitly requests a server cursor, server cursors are not created for statements such as INSERT.

Server cursors cannot be used with statements that generate more than one

recordset.

This restriction applies to all statements described in Generating Multiple Recordsets. For more information, see <u>Generating Multiple Recordsets</u>. If a server cursor is used with any statement that generates multiple recordsets, an application can return one of the following errors:

- Cannot open a cursor on a stored procedure that has anything other than a single SELECT statement in it.
- sp_cursoropen. The statement parameter can only be a single SELECT statement or stored procedure.

This example shows the opening of a dynamic server cursor:

Dim rs As New ADODB.Recordset

• • •

rs.Open "SELECT * FROM titles", , adOpenDynamic, adLockOptimis rs.Close

See Also

API Server Cursors

Default Result Sets

Scrolling and Retrieving Rows

An application can use the **MoveFirst**, **MoveLast**, **MoveNext**, and **MovePrevious** methods to scroll through a recordset to retrieve rows. Use the **MoveFirst** method to move the current record position to the first record in the **Recordset**. Use the **MoveLast** method to move the current record position to the last record in the **Recordset**.

Use the **MoveNext** method to move the current record position one record forward. If the last record is the current record and you call the **MoveNext** method, ADO sets the current record to the position after the last record in the **Recordset** and sets the EOF property to **True**. An attempt to move forward when the **EOF** property is set to **True** generates an error.

Use the **MovePrevious** method to move the current record position one record backward. If the first record is the current record and you call the **MovePrevious** method, ADO sets the current record to the position before the first record in the **Recordset** and sets the BOF property to **True**. An attempt to move backward when the **BOF** property is set to **True** generates an error.

If the **Recordset** object does not support backward cursor movement, a call to the **MoveFirst** or **MovePrevious** methods generates an error. For example, the default setting of the **CursorType** property is **adOpenForwardOnly**, which supports only the **MoveLast** and **MoveNext** methods.

Determining Recordset Limits

An application can use the **BOF** and **EOF** properties to determine whether a **Recordset** object contains records or whether you have gone beyond the limits of a **Recordset** object when you move from record to record. By testing the values of the **BOF** and **EOF** properties, an application can avoid generating an error by using the **MoveFirst**, **MoveLast**, **MoveNext** and **MovePrevious** methods.

The **BOF** property returns **True** (-1) if the current record position is before the first record, and returns **False** (0) if the current record position is on or after the first record. The **EOF** property returns **True** if the current record position is after the last record, and returns **False** if the current record position is on or before the

last record. If the **BOF** and **EOF** properties both are set to **True**, there is no current record. In this situation, the **RecordCount** property is set to zero.

If you delete the last remaining record in the **Recordset** object, the **BOF** and **EOF** properties may remain **False** until you attempt to reposition the current record.

Bookmarking Rows

An application can use the **Bookmark** property to save the position of the current record and to return to that record at any time. When you open a **Recordset** object, each of its records has a unique bookmark. To save the bookmark for the current record, assign the value of the **Bookmark** property to a variable. To return to that record at any time after moving to a different record, set the **Recordset** object **Bookmark** property to the value of that variable.

The user may not be able to view the value of the bookmark. Also, users should not expect bookmarks to be directly comparable; two bookmarks that refer to the same record may have different values.

If you use the **Clone** method to create a copy of a **Recordset** object, the **Bookmark** property settings for the original and for the duplicate **Recordset** objects are identical and you can use them interchangeably. However, you cannot use bookmarks from different **Recordset** objects interchangeably, even if they were created from the same source or command.

Performing Transactions in ADO

ADO supports transaction management in Microsoft® SQL Server[™] 2000, allowing an application to perform explicitly and implicitly started transactions on a single connection to an instance of SQL Server. After the connection is established, a recordset is opened on the result set of a select query, using a dynamic cursor and pessimistic locking (properties of a **Recordset** object). After you edit or update the data, you select whether to commit the changes or cancel them. The data changed in the transaction can then be committed or rolled back.

To perform an explicit transaction in an application

- 1. Open a new connection to an instance of SQL Server.
- 2. Retrieve a recordset from an instance of SQL Server.
- 3. Call the **BeginTrans** method of the **Connection** object to begin the transaction.
- 4. Make changes to the recordset.
- 5. Call the **CommitTrans** method of the **Connection** object to save changes to the recordset

Or

Call the **RollbackTrans** method of the **Connection** object to discard changes to the recordset.

Managing a Transaction

This example shows how to use the ADO transaction methods **BeginTrans**, **CommitTrans**, and **RollbackTrans** to manage a transaction.

Dim cn As New ADODB.Connection

Dim rs As New ADODB.Recordset

... ' Open connection. cn.Open

'Open titles table.
rs.Open "SELECT * FROM titles", Cn, adOpenDynamic, adLockPessi
'Begin the transaction.
rs.MoveFirst
cn.BeginTrans
'User loops through the recordset making changes.

...
'Ask if the user wants to commit all the changes made.
If MsgBox("Save all changes?", vbYesNo) = vbYes Then cn.CommitTrans

cn.RollbackTrans End If

See Also

Transactions

Handling Errors and Messages in ADO

ADO applications use the **Errors** collection and the **Error** object to return provider-specific error information to an application. The **Errors** collection contains the errors generated by a single operation. Each **Error** object constitutes one such error in the collection. To get information about an error, query the properties of an **Error** object from the **Connection** object. To get all the **Error** objects in the **Errors** collection, use code to loop through the collection.

ADO errors (for example, invalid use of ADO properties or methods), as opposed to provider errors, do not appear in the **Errors** collection. ADO errors are captured by the exception handling mechanism of your run-time environment. For example, in Microsoft® Visual Basic®, the occurrence of an ADO error triggers an **On Error** event and appears as a Visual Basic **Error** object.

If you want to trap both provider-specific errors (by querying the properties of an **Error** object) and ADO errors (by trapping ADO errors through the run-time exception handler) in your application, you have to write error-handling code for both. For more information about ADO Error Codes, see the MSDN Library at <u>Microsoft Web site</u>.

Warning messages that do not stop code execution can be saved in the **Errors** collection. A warning message has a positive number value, which differentiates it from an error message.

However, critical warning or status messages (such as calls made with unsupported or conflicting properties) may be ignored by ADO and not saved to the **Errors** collection if the operation succeeded.

The properties of an **Error** object contain specific details about each error:

- The **Description** property contains the text of the error.
- The **Number** property contains the long integer value of the error constant.

- The **Source** property identifies the object that raised the error.
- The **SQLState** and **NativeError** properties provide information from SQL data sources.
- The **HelpFile** and **HelpContext** properties indicate the appropriate Microsoft Windows[®] Help file and topic, respectively, (if any exist) for the error.

This code fragment, taken from the ADO Introductory Visual Basic Sample, shows how to create a basic data provider error log. The code enumerates the first five properties (all properties except for **HelpFile** and **HelpContext**) of each **Error** object in the **Errors** collection and displays them in a list on a Visual Basic form. In this example, the variable *errLoop* is an **Error** object in the **Errors** collection. The variable *strError* is an array of five strings, with each array element corresponding to a label and a specific property of an **Error** object. The routine loops through each **Error** object, exposes the value for each specified property, and displays the results as items in a list. The routine provides a count of the errors, using the **Errors** collection **Count** property, and clears out the **Errors** collection (using the **Clear** property).

Private Sub ErrorLog()

```
...
Dim errLoop As ADODB.Error
```

```
' Loop through each Error object in Errors collection.
For Each errLoop In cn.Errors
```

```
Dim strError(5)
Dim i As Integer
```

```
strError(0) = "Error Number: " & errLoop.Number
strError(1) = " Description: " & errLoop.Description
strError(2) = " Source: " & errLoop.Source
strError(3) = " SQL State: " & errLoop.SQLState
```

```
strError(4) = " Native Error: " & errLoop.NativeError
' Loop through the five specified properties of Error object.
i = 0
Do While i < 5
Form2.lstErrors.AddItem strError(i)
i = i + 1
Loop</pre>
```

Form2.lstErrors.AddItem ""

Next

```
' Create string for summary count of errors.
c = cn.Errors.Count & " provider error(s) occurred."
```

' Display a count of the provider errors. Form2.lstErrors.AddItem c Form2.lstErrors.AddItem ''''

```
' Clear the Errors collection.
cn.Errors.Clear
```

See Also

ADO Connection and Error Handling

Handling Data Definition Language

Data definition language (DDL) statements are SQL statements that support the definition or declaration of database objects (for example, CREATE TABLE, DROP TABLE, and ALTER TABLE).

You can use the ADO **Command** object to issue DDL statements. To differentiate DDL statements from a table or stored procedure name, set the **CommandType** property of the **Command** object to **adCmdText**. Because executing DDL queries with this method does not generate any recordsets, there is no need for a **Recordset** object.

Microsoft[®] SQL Server[™] 2000 provides a group of query processing options that can be specified by using the SET statement. These SET options do not generate result sets and can be treated as the same category of DDL queries.

This example shows the use of the **Command** object to turn off the SET NOCOUNT option of the Transact-SQL SET statement. This example drops a table, creates a table, and then inserts data into the new table by using the **Execute** method of the **Command** object. **Recordset** objects are not created for this type of query. The **ADOTestTable** table may not exist in the database, so execution of DROP TABLE ADOTestTable may generate an error indicating the table does not exist in the database. Some error handling code is provided for this situation. The SET NOCOUNT ON SET option is also executed.

Dim Cn As New ADODB.Connection Dim Cmd As New ADODB.Command

' If the ADOTestTable does not exist, go to AdoError. On Error GoTo AdoError

' Connect using the SQLOLEDB provider. cn.Provider = "sqloledb" cn.Properties("Data Source").Value = "MyServerName" cn.Properties("Initial Catalog").Value = "northwind" cn.Properties("Integrated Security").Value = "SSPI" cn.Open

' Set up command object. Set Cmd.ActiveConnection = Cn Cmd.CommandText = "DROP TABLE ADOTestTable" Cmd.CommandType = adCmdText Cmd.Execute

Done:

Cmd.CommandText = "SET NOCOUNT ON" Cmd.Execute Cmd.CommandText = "CREATE TABLE ADOTestTable (id int, nai Cmd.Execute Cmd.CommandText = "INSERT INTO ADOTestTable values(1, 'Jai Cmd.Execute Cn.Close Exit Sub

AdoError: Dim errLoop As Error

Dim strError As String

'Enumerate Errors collection and display properties of
'each Error object.
Set Errs1 = Cn.Errors
For Each errLoop In Errs1

Debug.Print errLoop.SQLState
Debug.Print errLoop.NativeError
Debug.Print errLoop.Description

Next

GoTo Done

End Sub

Using ADOX

Microsoft® ActiveX® Data Objects Extensions for Data Definition Language and Security (ADOX) is an extension to the ADO objects and programming model. ADOX includes objects for schema creation and modification, as well as security. However, certain features of ADOX are not be supported by the Microsoft SQL Server OLE DB Provider (SQLOLEDB). For more information, see <u>Provider Support for ADOX</u>.

Managing Long Data Types

Long data types include **ntext**, **text**, and **image** data types. **ntext**, **text**, and **image** data can be so large that they cannot be retrieved in one operation or fit into memory. If the long data can fit into memory, the **Value** property of the **Field** object can be used to retrieve all the data in one operation. If the long data is too large to fit into memory, the data must be retrieved or written in chunks. You can manipulate long data in chunks through the **Field** object or through the **Parameter** object.

The **Field** object allows you to write and read long data through the **Recordset** object. The **AppendChunk** method of the **Field** object allows you to append data at the end of the current data when the query has already been executed. The **GetChunk** method allows you to read the data in chunks.

With the **Parameter** object, there is no **GetChunk** method, and there is no **Recordset** object when you are dealing with long data at run time. With the **Parameter** object, long data is bound at run time and executed with the **Command** object.

There are some restrictions for long data when using MSDASQL. If no server cursor is used, all long columns must be to the right of all nonlong columns. If there are multiple long columns, the long columns must be accessed in order (from left to right).

This example shows how to use ADO with SQLOLEDB to read and write **image** data. The critical routines are the while loops that copy the long data (**image**) to a variable and write the variable to a record in chunks (using the **GetChunk** and **AppendChunk** methods).

Before setting up the destination table in this example, make sure to run the **sp_dboption** stored procedure:

EXEC sp_dboption 'pubs', 'Select into/bulkcopy', 'True'

The destination table is a copy of the **pub_info** table in the **pubs** database. Create the table by running:

USE pubs

SELECT * INTO pub_info_x FROM pub_info GO

The **pub_info_x** table is the destination table in which the long data will be inserted.

The ADO code is:

Public Sub AppendChunkX()

Dim cn As ADODB.Connection Dim rstPubInfo As ADODB.Recordset Dim strCn As String Dim strPubID As String Dim strPRInfo As String Dim lngOffset As Long Dim lngLogoSize As Long Dim varLogo As Variant Dim varChunk As Variant

Const conChunkSize = 100

'Open a connection.
Set cn = New ADODB.Connection
strCn = "Server=srv;Database=pubs;UID=sa;Pwd=;"

cn.Provider = "sqloledb" cn.Open strCn

'Open the pub_info_x table. Set rstPubInfo = New ADODB.Recordset rstPubInfo.CursorType = adOpenDynamic rstPubInfo.LockType = adLockOptimistic rstPubInfo.Open "pub_info_x", cn, , , adCmdTable

```
'Prompt for a logo to copy.
strMsg = "Available logos are : " & vbCr & vbCr
```

```
Do While Not rstPubInfo.EOF
strMsg = strMsg & rstPubInfo!pub_id & vbCr & _
Left(rstPubInfo!pr_info,
InStr(rstPubInfo!pr_info, ",") - 1) & vbCr & vbCr
rstPubInfo.MoveNext
Loop
```

```
strMsg = strMsg & "Enter the ID of a logo to copy:"
strPubID = InputBox(strMsg)
```

```
' Copy the logo to a variable in chunks.
rstPubInfo.Filter = "pub_id = "" & strPubID & """
lngLogoSize = rstPubInfo!logo.ActualSize
Do While lngOffset < lngLogoSize
varChunk = rstPubInfo!logo.GetChunk(conChunkSize)
varLogo = varLogo & varChunk
lngOffset = lngOffset + conChunkSize
Loop
```

```
' Get data from the user.
strPubID = Trim(InputBox("Enter a new pub ID:"))
strPRInfo = Trim(InputBox("Enter descriptive text:"))
```

```
' Add a new record, copying the logo in chunks.
rstPubInfo.AddNew
rstPubInfo!pub_id = strPubID
rstPubInfo!pr_info = strPRInfo
lngOffset = 0 ' Reset offset.
```

```
Do While lngOffset < lngLogoSize
varChunk = LeftB(RightB(varLogo, lngLogoSize - _
lngOffset),conChunkSize)
rstPubInfo!logo.AppendChunk varChunk
lngOffset = lngOffset + conChunkSize
Loop
```

rstPubInfo.Update

' Show the newly added data. MsgBox "New record: " & rstPubInfo!pub_id & vbCr & _ "Description: " & rstPubInfo!pr_info & vbCr & _ "Logo size: " & rstPubInfo!logo.ActualSize

rstPubInfo.Close cn.Close

End Sub

See Also

ADO and Long Data Types (C++) ADO and Long Data Types (Visual Basic) ADO and Long Data Types (Web)

ADO Support for SQL Server XML Features

ADO applications can use the Microsoft OLE DB Provider for SQL Server (SQLOLEDB) to use the XML features of Microsoft® SQL Server[™] 2000. Applications can:

- Use template queries. A template is a valid XML document, containing one or more SQL queries.
- Use XML Views on the database. XML Views provide a mapping from an XML document to a relational database. The mapping is done by annotating an XML-Data Reduced Schema. Once the XML View is defined, an XPath query can be executed to retrieve data from the database.
- Use the OpenXML extension to Transact-SQL. OpenXML provides a relational view on an XML document by allowing stored procedures to process XML and generate rowsets from the data for use by Transact SQL statements.

XML-Related Properties

The Microsoft OLE DB Provider for SQL Server (SQLOLEDB) implements several new provider-specific properties that are used to retrieve XML from Microsoft® SQL Server[™] 2000. These properties are available to ADO applications as dynamic properties.

These properties are used to specify the mapping schema against which an XPath query is specified as a command or to specify an XSL file to process the results.

Property Name	Description	
Base Path Property	The Base Path property specifies a file	
	path or URL to use for resolving relative	
	paths in a template (for example, XSL on a	
	template root directory, sql:mapping-	
	schema attribute on a sql:xpath:query,	
	external schema references in an inline	
	schema, or Mapping Schema and XML	
	Root properties).	
Content Type Property	The Content Type property returns the	
	output content type of an XML	
	transmission.	
Mapping Schema Property	The Mapping Schema property specifies a	
	file name or URL that points to the	
	mapping schema used by the provider to	
	translate an XPath command.	
SS STREAM FLAGS Property		
	specifies how an application manages	
	mapping schemas, XSL files, and	
	templates.	
XML Root Property	The XML Root property provides a root	
	tag in which the query result is wrapped to	
	return a well-formed document.	
XSL Property	The XSL property specifies an XSL file	

name or URL applied to the result of a
query.

These ADO properties map to standard OLE DB 2.6 properties and are used when retrieving the results of a **Command** execution as a stream.

Property Name	Description	
Output Encoding Property	The Output Encoding property specifies	
	the encoding to use in the stream set or	
	returned by the Execute method.	
Output Stream Property	The Output Stream property specifies the	
	stream containing the results returned by	
	the Execute method.	

Using Streams for Command Input

ADO queries can be specified by setting the **CommandText** property on the **Command** object or by associating the stream with the **Command** object using the **CommandStream** property. This example demonstrates using a stream to access the **Northwind** database. It uses an Active Server Page (ASP) and is written in Microsoft Visual Basic® Scripting Edition.

Using XML Template Queries in Streams

The application initializes the ADO **Stream** object to contain query text:

```
Dim adoStreamQuery
Set adoStreamQuery = Server.CreateObject("ADODB.Stream")
adoStreamQuery.Open
```

The application requires a reference to the XML Namespace identified by the sql: prefix of the <sql:query> tag. The SELECT statement with a reference to the sql: Namespace takes this form:

```
<ROOT xmlns:sql='urn:schemas-microsoft-com:xml-sql'>
<sql:query> SELECT * FROM PRODUCTS ORDER BY PRODUCT
</ROOT>
```

By using the FOR XML AUTO mode of the SELECT statement, this query requests that results are returned in XML format, rather than as a **Recordset** object. For more information, see <u>Retrieving and Writing XML Data</u>.

The command is then assigned to a string variable, and copied to the adoStreamQuery stream, which is associated with an ADO **Command** object:

```
sQuery = "<ROOT xmlns:sql='urn:schemas-microsoft-com:xml-sql'>
<sql:query> SELECT * FROM PRODUCTS ORDER BY PRODUCT
</ROOT>"
adoStreamQuery.WriteText sQuery, adWriteChar
adoStreamQuery.Position = 0
```

```
Dim adoCmd
Set adoCmd = Server.CreateObject("ADODB.Command")
Set adoCmd.CommandStream = adoStreamQuery
```

Setting the Command Language Dialect

The second requirement of the application is setting the command language dialect, which specifies how the Microsoft OLE DB Provider for SQL Server interprets the command text received from ADO. The dialect is specified by a globally unique identifier (GUID) and is set using the **Dialect** property of the **Command** object. The Microsoft OLE DB Provider for SQL Server (SQLOLEDB) supports these values.

ADO Value	OLE DB Constant	Description
{C8B521FB-5CF3-11CE-	DBGUID_DEFAULT	Provider-specific
ADE5-00AA0044773D}		default behavior
{C8B522D7-5CF3-11CE-	DBGUID_SQL	Transact-SQL query
ADE5-00AA0044773D}		
{5D531CB2-E6Ed-11D2-	DBGUID_MSSQLXML	XML template
B252-00C04F681B71}		query
{EC2A4293-E898-11D2-	DBGUID_XPATH	XPath query
B1B7-00C04F680C56}		

The command dialect for XML queries is specified as follows:

AdoCmd.Dialect = "{5D531CB2-E6Ed-11D2-B252-00C04F681B71}'

Retrieving Result Sets into Streams

In addition to receiving results in a **Recordset** object, an ADO application can use the **Stream** object to contain these results in XML format. These results also can be streamed into any object that supports the OLE DB **IStream** interface, (for example, the ASP **Response** object.

This example demonstrates using a stream to access the **Northwind** database. It uses an Active Server Page (ASP) and is written in Microsoft Visual Basic Scripting Edition (VBScript).

FOR XML Queries

The FOR XML clause, which allows SQL Server to return data in the form of an XML document, has been added to the SELECT statement in SQL Server 2000. The syntax of the FOR XML clause is:

FOR XML [RAW|AUTO|EXPLICIT]

FOR XML RAW generates generic row elements with column values as attributes. FOR XML AUTO uses heuristics to generate a hierarchical tree with element names based on table names. FOR XML EXPLICIT provides complete control over the format of the XML returned by the query. For more information, see <u>Retrieving XML Data Using FOR XML</u>.

The command can be entered in the form of:

<ROOT xmlns:sql='urn:schemas-microsoft-com:xml-sql'> SELECT * FROM PRODUCTS ORDER BY PRODUCTNAME FOR

The command can also be entered in the form of a template query. When constructing a template query for use with the ADO **Command** object, the application must enclose the command text in <sql:query> </sql:query> tags to reference an XML Namespace specific to SQL Server queries. The command is entered in the form of:

<sql:query> SELECT * FROM PRODUCTS ORDER BY PRODUCTNAME FOR XML AUTO </sql:query> The application must also specify where to send the output of the query. When using the FOR XML clause, the application can specifies a **Stream** object to receive the resulting XML output. In this example, the application uses the ASP **Response** object by setting the **Output Stream** property on the ADO **Command** object:

adoCmd.Properties("Output Stream") = Response

After the output stream has been associated with the **Command** object using the **Output Stream** property, the command can be executed. The application sets the adExecuteStream parameter to retrieve results in the form of a stream instead as a record set, which is the default. This example encloses the stream in XML tags that create an XML data island.

```
Response.write "<XML ID='MyDataIsle'>"
adoCmd.Execute , , adExecuteStream
Response.write "</XML>"
```

At this point in the code execution, the application has streamed XML to the client browser and to display it using client-side VBScript to bind the XML document to an instance of the Document Object Model (DOM), looping through each child node to build a list of products in HTML:

<SCRIPT language="VBScript" For="window" Event="onload">

```
Dim xmlDoc
Set xmlDoc = MyDataIsle.XMLDocument
xmlDoc.resolveExternals=false
xmlDoc.async=false
```

Dim root, child Set root = xmlDoc.documentElement

```
For each child in root.childNodes
dim OutputXML
OutputXML = document.all("log").innerHTML
```

document.all("log").innerHTML = OutputXML & "" & child.ge
Next

```
</SCRIPT>
</HEAD>
<BODY>
<H3>Client-side processing of XML Document MyDataIsle</H3>
<UL id=log>
</UL>
</BODY>
</HTML>
```

Example

This is the complete code listing from the ASP described previously. The ASP:

- Queries SQL Server 2000.
- Binds the resulting XML stream to the DOM.
- Displays data from several nodes.

```
<%@ LANGUAGE = VBScript %>
<% Option Explicit %>
```

```
<HTML>
<HEAD>
<META NAME="GENERATOR" Content="Microsoft Developer Stuc
<META HTTP-EQUIV="Content-Type" content="text/html" charset='
<TITLE>ADO 2.6 E</TITLE>
```

```
<!-- #include file="adovbs.inc" -->
```

<%

Response.Write "<H3>Server-side processing</H3>"

```
Dim adoConn
Set adoConn = Server.CreateObject("ADODB.Connection")
```

```
Dim sConn
sConn = "Provider=SQLOLEDB;Data Source=MYSERVER1;Initial C
adoConn.ConnectionString = sConn
adoConn.CursorLocation = adUseClient
adoConn.Open
```

```
Dim adoCmd
Set adoCmd = Server.CreateObject("ADODB.Command")
Set adoCmd.ActiveConnection = adoConn
```

```
Dim sQuery
sQuery = "<ROOT xmlns:sql='urn:schemas-microsoft-com:xml-sql'><
```

```
Dim adoStreamQuery
Set adoStreamQuery = Server.CreateObject("ADODB.Stream")
adoStreamQuery.Open
adoStreamQuery.WriteText sQuery, adWriteChar
adoStreamQuery.Position = 0
```

```
Set adoCmd.CommandStream = adoStreamQuery
adoCmd.Dialect = "{5D531CB2-E6Ed-11D2-B252-00C04F681B71]
```

```
Response.write "Pushing XML to client for processing " & "<BR/>"
```

```
adoCmd.Properties("Output Stream") = Response
Response.write "<XML ID='MyDataIsle'>"
adoCmd.Execute , , adExecuteStream
Response.write "</XML>"
%>
```

<SCRIPT language="VBScript" For="window" Event="onload">

```
Dim xmlDoc
Set xmlDoc = MyDataIsle.XMLDocument
xmlDoc.resolveExternals=false
xmlDoc.async=false
```

```
Dim root, child
Set root = xmlDoc.documentElement
```

```
For each child in root.childNodes
dim OutputXML
OutputXML = document.all("log").innerHTML
document.all("log").innerHTML = OutputXML & "<LI>" & child.
Next
```

```
</SCRIPT>
</HEAD>
<BODY>
<H3>Client-side processing of XML Document MyDataIsle</H3>
<UL id=log>
</UL>
</BODY>
</HTML>
```

See Also

ADO and FOR XML

Mapping an XML Schema to a Relational Schema Using Annotated Schemas

An application can create XML views of relational data using annotated XDR (XML-Data Reduced) schemas, which can then be queried using XPath queries. This process is conceptually similar to creating views using Transact-SQL CREATE VIEW statements, and then specifying SQL queries against the view. Annotated schemas, which are Microsoft-developed extensions to the XML Data specification, allow client applications to view a relational database as an XML document instead of a group of tables. An XML file that maps XML elements and attributes to tables and columns of a relational database is called a Mapping Schema. Applications can use these two technologies to query Microsoft® SQL Server[™] without using SQL commands and without knowing the relational design of the database. For more information about XPath queries, see <u>Using XPath Queries</u>. For more information about XML views and annotated schemas, see <u>Creating XML Views Using Annotated XDR Schemas</u>.

The following example demonstrates how to build an XPath query that is functionally equivalent to this Transact-SQL statement:

SELECT o.OrderID, o.OrderDate from Orders o, Customers c, WHERE o.CustomerID = c.CustomerID and c.CompanyName = ?

This example passes the CompanyName, Tortuga Restaurante, as an input parameter.

The **Customers** and **Orders** tables from the **Northwind** database are used to create a mapping schema. This is the structure of the **Customers** and **Orders** tables, including primary and foreign key relationships.

CREATE TABLE [Customers] [CustomerID] [nchar] (5) NOT NULL , [CompanyName] [nvarchar] (40) NOT NULL , [ContactName] [nvarchar] (30) NULL , [ContactTitle] [nvarchar] (30) NULL , [Address] [nvarchar] (60) NULL , [City] [nvarchar] (15) NULL , [Region] [nvarchar] (15) NULL , [PostalCode] [nvarchar] (10) NULL , [Country] [nvarchar] (15) NULL , [Phone] [nvarchar] (24) NULL , [Fax] [nvarchar] (24) NULL

PRIMARY KEY [CustomerID]

CREATE TABLE [Orders] ([OrderID] [int] IDENTITY (1, 1) NOT NULL, [CustomerID] [nchar] (5) NULL, [EmployeeID] [int] NULL, [OrderDate] [datetime] NULL, [RequiredDate] [datetime] NULL, [ShippedDate] [datetime] NULL, [ShipVia] [int] NULL, [Freight] [money] NULL, [Freight] [money] NULL, [ShipName] [nvarchar] (40) NULL, [ShipAddress] [nvarchar] (60) NULL, [ShipAddress] [nvarchar] (60) NULL, [ShipCity] [nvarchar] (15) NULL, [ShipRegion] [nvarchar] (15) NULL, [ShipPostalCode] [nvarchar] (10) NULL, [ShipCountry] [nvarchar] (15) NULL

PRIMARY KEY [OrderID] FOREIGN KEY [Customers].[CustomerID]

The example SQL query requires the **OrderID**, **OrderDate**, and **CustomerID** columns from the **Orders** table, and the **CompanyName** and **CustomerID** columns from the **Customers** table.

The application also requires a mapping schema, which in this example, is stored in the Orders.xml file in the virtual root directory.

The document contains Namespace declarations, specifically the XML-Data namespace. These table-mapping elements are also included:

- The sql:relation attribute, which is used to identify the table or view in the database. Inside each element are attributes that map to columns in the table identified by the element.
- The sql:field attribute, which is used to identify the field in the SQL table.
- The sql:relationship attribute, which is used to identify the primary and foreign key relationships between the two tables.

```
<?xml version="1.0" ?>
```

```
<Schema xmlns="urn:schemas-microsoft-com:xml-data"
xmlns:dt="urn:schemas-microsoft-com:datatypes"
xmlns:sql="urn:schemas-microsoft-com:xml-sql">
```

```
<ElementType name="Order" sql:relation="Orders" >
```

```
<AttributeType name="CustomerID" />
```

```
<AttributeType name="OrderID" />
```

```
<AttributeType name="OrderDate" />
```

```
<attribute type="CustomerID" sql:field="CustomerID" />
```

```
<attribute type="OrderID" sql:field="OrderID" />
```

```
<attribute type="OrderDate" sql:field="OrderDate" />
```

</ElementType>

```
<ElementType name="Customer" sql:relation="Customers" >
<AttributeType name="CustomerID" />
<AttributeType name="CompanyName" />
<attribute type="CustomerID" sql:field="CustomerID" />
<attribute type="CompanyName" sql:field="CompanyName" />
<element type="Order" >
<sql:relationship key-relation="Customers" key="CustomerID"
```

```
foreign-key="CustomerID" foreign-relation="Orders" />
</element>
</ElementType>
</Schema>
```

Using an Active Server Page (ASP), an application user generates a URL containing a company name for which he or she wants to see orders. In this example, the URL takes the form:

```
http://WebServer/Vroot/Orders.asp?CompanyName="Tortuga%20Rest
```

Using the customer name passed in by the user, the ASP constructs this XPath query to run against the mapping schema:

```
Customer[@CompanyName="Tortuga Restaurante"]
```

This query string is passed to the ADO **Command** object and executed, returning the results in an XML stream.

The ASP begins by using the ASP **Request** object to capture the CompanyName passed in using the URL and storing it in a string variable called sCompanyName.

```
dim sCompanyName
sCompanyName = Request.QueryString("CompanyName")
```

The application then creates ADO **Connection** and **Command** objects. Because the application issues commands written as XPATH queries, it must use the XPATH command dialect.

```
adoCmd.CommandText = "Customer[@CompanyName=" & sCompar
adoCmd.Dialect = "{ec2a4293-e898-11d2-b1b7-00c04f680c56}"
```

The application then sets properties specific to the Microsoft OLE DB Provider for SQL Server: **Mapping Schema** and **Base Path**. The application sets the **Mapping Schema** property to the name of the mapping schema file, and **Base Path** property to the directory containing the mapping schema file.

```
adoCmd.Properties("Mapping Schema") = "Orders.xml"
```

adoCmd.Properties("Base Path") = "C:\INETPUB\WWWROOT\Kov

After the Output Stream property is set to the ASP **Response** object, the command can be executed. The application sets the adExecuteStream parameter of the **Command** object, and encloses the setting in XML tags to create an XML data island.

```
Response.write "<XML ID='MyDataIsle'>"
adoCmd.Execute , , adExecuteStream
Response.write "</XML>"
%>
```

At this point in the code execution, the application has passed the XML stream to the client browser. The XML stream is displayed using client-side VBScript to bind the XML document to an instance of the DOM, and by looping through each child node to build a list of OrderIDs and OrderDates using HTML.

Examples

This is the complete code listing from the ASP described previously.

```
<HTML>
<HEAD>
<META NAME="GENERATOR" Content="Microsoft Developer Stuc
<META HTTP-EQUIV="Content-Type" content="text/html" charset='
<TITLE>XPATH Query Annotated Schema Orders.asp</TITLE>
```

```
<STYLE>
BODY
{
FONT-FAMILY: Tahoma;
FONT-SIZE: 8pt;
OVERFLOW: auto
}
H3
```

```
{
  FONT-FAMILY: Tahoma;
  FONT-SIZE: 8pt;
  OVERFLOW: auto
}
```

</STYLE>

```
<!-- #include file="adovbs.inc" -->
<%
dim sCompanyName
sCompanyName = Request.QueryString("CompanyName")
If Len(sCompanyName) = 0 then
Response.redirect "http://MYSERVER1/Kowalski/OrdersErr.asp"
Else
Dim sConn
sConn = "Provider=SQLOLEDB; Data Source=MYSERVER1; Initial
User ID=SA;Password=;"
```

```
Dim adoConn
Set adoConn = Server.CreateObject("ADODB.Connection")
adoConn.ConnectionString = sConn
adoConn.CursorLocation = adUseClient
adoConn.Open
```

```
Dim adoCmd
Set adoCmd = CreateObject("ADODB.Command")
Set adoCmd.ActiveConnection = adoConn
adoCmd.CommandText = "/Customer[@CompanyName=" & sC
adoCmd.Dialect = "{ec2a4293-e898-11d2-b1b7-00c04f680c56}"
```

```
adoCmd.Properties("Mapping Schema") = "Orders.xml"
adoCmd.Properties("Base Path") = "C:\INETPUB\WWWROOT
```

adoCmd.Properties("Output Stream") = Response

```
Response.write "<XML ID='MyDataIsle'>"
adoCmd.Execute , , adExecuteStream
Response.write "</XML>"
End If
%>
```

<SCRIPT language="VBScript" For="window" Event="onload">

```
Dim xmlDoc
Set xmlDoc = MyDataIsle.XMLDocument
xmlDoc.resolveExternals=false
xmlDoc.async=false
```

```
Dim root, child, header, OutputHeader
Set root = xmlDoc.documentElement
```

```
OutputHeader = document.all("header").innerHTML
OutputHeader = OutputHeader & "CustomerID: " & root.getAttribut
document.all("header").innerHTML = OutputHeader
```

```
For each child in root.childNodes
dim OutputOrders, OrderList
OutputOrders = document.all("Orders").innerHTML
OrderList = "<LI> Order # " & child.getAttribute("OrderID") & ",
child.getAttribute("OrderDate") & "</LI>"
TotalPage = OutputOrders & OrderList
document.all("Orders").innerHTML = TotalPage
Next
```

</SCRIPT>

```
</HEAD>
<BODY>
<H3>Client-side processing of XML Document MyDataIsle</H3>
<DIV id=Header></DIV>
<UL id=Orders></UL>
</BODY>
</HTML>
```

See Also

ADO and XPath Query

ADO Support for OpenXML

OpenXML is a SQL Server 2000 extension to Transact-SQL that allows stored procedures to process XML and generate rowsets from the data for use by Transact-SQL statements. In the following example, ADO passes an XML document to a stored procedure. The stored procedure executes a SELECT statement generating a rowset. This rowset can then be processed by the stored procedure, or returned to the client as an ADO **Recordset**.

To use stored procedures to process XML:

- 1. Execute the **sp_xml_preparedocument** stored procedure to prepare the XML document for use by Transact-SQL statements.
- 2. Use the OpenXML-generated rowset in one or more queries.
- 3. Execute **sp_xml_removedocument** to remove the prepared XML document from memory.

The ASP calls **Command.Execute** to execute the stored procedure, and passes in the XML document. The application then executes

sp_xml_preparedocument to create an in-memory representation of the XML document. **sp_xml_preparedocument** has an output parameter (@*iDoc*, **int**), which is a pointer to the prepared XML document, and an input parameter (@*XMLDoc*, **VarChar**(2000), which contains the text of an XML document to be accessed using T-SQL statements.

EXECUTE sp_xml_preparedocument @iDoc OUTPUT, @XMLD

In this SELECT statement, the application passes in the *@iDoc* handle, an XPath command '/Root/Customers', a flag '1' indicating that the XML is attribute-centric, and a WITH clause describing the structure of the rowset to be returned.

SELECT * FROM OpenXML(@iDoc, '/ROOT/Customers',1) WITH (CustomerID varchar(10), ContactName varchar(20)) Any Transact-SQL statement that operates with a rowset can be used with the OpenXML keyword. For example, an application can also use INSERT, UPDATE, DELETE, and JOIN statements.

After the application completes processing of the in-memory XML document, it releases the document by passing the *@iDoc* parameter to **sp_xml_removedocument**:

EXECUTE sp_xml_removedocument @iDoc

For more information about OpenXML, see <u>Writing XML Using OpenXML</u>.

Example

This is the complete listing of the stored procedure discussed previously.

```
CREATE PROCEDURE SP_OpenXML_Example
@XMLDoc varchar(2000)
```

AS

DECLARE @ReturnCode INT DECLARE @iDoc int

EXECUTE sp_xml_preparedocument @iDoc OUTPUT, @XMLD

```
SELECT * FROM OpenXML(@iDoc, '/ROOT/Customers',1)
WITH (CustomerID varchar(10), ContactName varchar(20))
```

EXECUTE sp_xml_removedocument @iDoc

```
SELECT @ReturnCode = 1
RETURN @ReturnCode
GO
```

Active Server Page

This is the complete listing of the ASP discussed previously.

```
<HTML>
<HEAD>
<META NAME="GENERATOR" Content="Microsoft Developer Stuc
<META HTTP-EQUIV="Content-Type" content="text/html" charset='
<TITLE>ADO 2.6 OpenXML Example - OpenXML.asp</TITLE>
```

```
<STYLE>
BODY
{
FONT-FAMILY: Tahoma;
FONT-SIZE: 8pt;
OVERFLOW: auto
}
H3
```

```
{
  FONT-FAMILY: Tahoma;
  FONT-SIZE: 8pt;
  OVERFLOW: auto
}
```

```
</STYLE>
```

```
<!-- #include file="adovbs.inc" -->
<%
Response.Write "Page Generated @ " & Now() & "<BR/>"
```

```
Dim sConn
sConn = "Provider=SQLOLEDB;Data Source=MYSERVER1;Initial
```

Response.write "Connect String = " & sConn & "
"

Dim adoConn

```
Set adoConn = Server.CreateObject("ADODB.Connection")
adoConn.ConnectionString = sConn
adoConn.CursorLocation = adUseClient
adoConn.Open
```

Response.write "ADO Version = " & adoConn.Version & "
" Response.write "adoConn.State = " & adoConn.State & "
"

Dim sXMLDoc, sQuery

```
sXMLDoc = "<ROOT>"
sXMLDoc = sXMLDoc & "<Customers CustomerID='VINET' Cont
sXMLDoc = sXMLDoc & "<Orders CustomerID='VINET' Employe
sXMLDoc = sXMLDoc & "<Order_0020_Details OrderID='10248' ]
sXMLDoc = sXMLDoc & "<Order_0020_Details OrderID='10248' ]
sXMLDoc = sXMLDoc & "</Orders>"
sXMLDoc = sXMLDoc & "</Customers>"
sXMLDoc = sXMLDoc & "<Customers CustomerID='LILAS' Conta
sXMLDoc = sXMLDoc & "<Orders CustomerID='LILAS' Employee
sXMLDoc = sXMLDoc & "<Order_0020_Details OrderID='10283' ]
sXMLDoc = sXMLDoc & "</Orders>"
sXMLDoc = sXMLDoc & "</Orders>"
sXMLDoc = sXMLDoc & "</Orders>"
```

sQuery = "SP_OpenXML_Example"
Response.write "sQuery = " & sQuery & "
"

Dim adoCmd Set adoCmd = Server.CreateObject("ADODB.Command") Set adoCmd.ActiveConnection = adoConn adoCmd.CommandText = sQuery adoCmd.CommandType = adCmdStoredProc adoCmd.Parameters.Refresh

adoCmd.Parameters.Item(1).Value = sXMLDoc

Dim adoRS
Set adoRS = adoCmd.Execute()

Response.write "Data = " & adoRS.Fields(0).Value & "
" adoRS.Close

Response.write "ReturnValue = " & adoCmd.Parameters.Item(0).Value

%> </HEAD> <BODY> </BODY> </HTML>

See Also

ADO and Open XML

SQL Server ADO Programmer's Reference

Microsoft® ActiveX® Data Objects (ADO) is a Win32 API used by applications to manipulate data in a database server through an OLE DB provider.

The SQL Server ADO Programmer's Reference does not document all of the ADO features. Those features that have provider-specific properties, parameters, or behaviors when used with the Microsoft OLE DB Provider for SQL Server (SQLOLEDB) are discussed.

SQLOLEDB allows ADO to access Microsoft SQL Server, and supports these provider-specific features:

- Command dynamic properties
- Connection dynamic properties
- Connection parameters
- Recordset dynamic properties

These features use ADO 2.6.

For more information about ADO, see the MSDN Library at the <u>Microsoft Web</u> <u>site</u>.

Objects

The ADO object model defines a collection of programmable objects that can be used by any of the Microsoft Visual languages (Microsoft® Visual Basic®, Microsoft Visual C++®, and Microsoft Visual J++TM); Web scripting languages such as Visual Basic Scripting Edition and Microsoft JScript®; and generally, any platform that supports both COM and Automation. The ADO object model, which contains nine objects and four collections, is designed to expose the most commonly used features of OLE DB.

The following diagram shows the relationships among the ADO objects and collections:

Connection	
Errors	Error
Command]
- Parameters	Parameter
- Recordset	
Fields	Field
Record	
Fields	Field
Stream	

The **Connection**, **Command**, and **Recordset** objects are the most commonly used ADO objects.

- The **Connection** object is used to establish connections between the client and database server.
- The **Command** object is used to issue commands, such as SQL queries and updates, to the database.
- The **Recordset** object is used to view and manipulate the results of the query.

Use the **Parameters** collection and **Parameter** objects when the query in the **Command** object requires parameters. The **Errors** collection and **Error** object are accessed through the **Connection** object after a provider error occurs. The **Fields** collection and **Field** object are accessed through the **Recordset** object after data exists in the **Recordset** object, and through the **Row** object.

The **Properties** collection provides information about the characteristics of the **Connection, Command, Recordset, Row, Field**, and **Stream** objects. Each **Property** object belonging to the **Properties** collection must be accessed through one of those six objects.

Connection
Command
Recordset
Record
Field
Stream
Properties Property

Although ADO defines an object hierarchy, all ADO objects except for the **Error**, **Field**, and **Property** objects can be created on their own. (This differs somewhat from the DAO and RDO object models, where an object must often be qualified with its parent objects when used.)

Because ADO offers flexibility in defining programmable objects, there are often several ways of accomplishing the same task. For example, to execute a query, you can use the **Execute** method of either the **Connection** object or the **Command** object.

See Also

Using the Fields Collection and Field Object

Using Parameters

Command Object

The **Command** object is used to query a database and return records in a **Recordset** object, to execute a bulk operation, or to manipulate the structure of a database.

In addition to the standard ADO properties, these dynamic properties are added to the **Properties** collection of the **Command** object.

Dynamic Properties

Access Order Property	Notification Granularity Property
Base Path Property	Notification Phases Property
Blocking Storage Objects Property	Objects Transacted Property
Bookmark Type Property	Others' Changes Visible Property
Bookmarkable Property	Others' Inserts Visible Property
Change Inserted Rows Property	Output Encoding Property
Column Privileges Property	Output Stream Property
Column Set Notification Property	Own Changes Visible Property
Content Type Property	Own Inserts Visible Property
Cursor Auto Fetch Property	Preserve on Abort Property
Defer Column Property	Preserve on Commit Property
Defer Prepare Property	Quick Restart Property
Delay Storage Object Updates	Reentrant Events Property
Property	
Fetch Backwards Property	Remove Deleted Rows Property
Hold Rows Property	Report Multiple Changes Property
IAccessor Property	Return Pending Inserts Property
IColumnsInfo Property	Row Delete Notification Property
IColumnsRowset Property	Row First Change Notification
	Property
IConnectionPointContainer	Row Insert Notification Property
<u>Property</u>	
IConvertType Property	Row Privileges Property

Immobile Rows Property	Row Resynchronization Notification Property
IRowset Property	Row Threading Model Property
IRowsetChange Property	Row Undo Change Notification
	Property
IRowsetIdentity Property	Row Undo Delete Notification
	Property
IRowsetInfo Property	Row Undo Insert Notification Property
IRowsetLocate Property	Row Update Notification Property
IRowsetResynch Property	Rowset Fetch Position Change
	Notification Property
IRowsetScroll Property	Rowset Release Notification Property
IRowsetUpdate Property	Scroll Backwards Property
ISequentialStream Property	Server Cursor Property
ISupportErrorInfo Property	Server Data on Insert Property
Literal Bookmarks Property	Skip Deleted Bookmarks Property
Literal Row Identity Property	SS STREAM FLAGS Property
Lock Mode Property	Strong Row Identity Property
Mapping Schema Property	Updatability Property
Maximum Open Rows Property	Use Bookmarks Property
Maximum Pending Rows Property	XML Root Property
Maximum Rows Property	XSL Property

See Also

Using the Command Object

Connection Object

A **Connection** object represents a unique session with a data source. In the case of a client/server database system, it may be equivalent to an actual network connection to the server.

In addition to the standard ADO properties, these dynamic properties are added to the **Properties** collection of the **Connection** object.

Active Sessions Property NULL Collation Order Property Asynchable Abort Property NULL Concatenation Behavior Property Asynchable Commit Property **OLE DB Version Property** Autocommit Isolation Levels **OLE Object Support Property** <u>Property</u> Catalog Location Property **Open Rowset Support Property ORDER BY Columns in Select List** Catalog Term Property <u>Property</u> **Column Definition Property Output Parameter Availability Property** Pass By Ref Accessors Property **Connect Timeout Property Current Catalog Property** Password Property **Data Source Property** Persist Security Info Property Data Source Name Property Persistent ID Type Property **Data Source Object Threading** Prepare Abort Behavior Property Model Property Prepare Commit Behavior Property **DBMS** Name Property **DBMS** Version Property Procedure Term Property Extended Properties Property Prompt Property Provider Friendly Name Property **GROUP BY Support Property** Heterogeneous Table Support Provider Name Property <u>Property</u>

Dynamic Properties

Identifier Case Sensitivity Property	Provider Version Property
Initial Catalog Property	Read-Only Data Source Property
Isolation Levels Property	Rowset Conversions on Command
	Property
Isolation Retention Property	Schema Term Property
Locale Identifier Property	Schema Usage Property
Maximum Index Size Property	SQL Support Property
Maximum Row Size Property	Structured Storage Property
Maximum Row Size Includes	Subquery Support Property
BLOB Property	
Maximum Tables in SELECT	Table Term Property
Property	
Multiple Parameter Sets Property	Transaction DDL Property
Multiple Results Property	User ID Property
Multiple Storage Objects Property	User Name Property
<u>Multi-Table Update Property</u>	Window Handle Property

Provider-Specific Connection Parameters

The Microsoft OLE DB Provider for SQL Server (SQLOLEDB), supports several provider-specific connection parameters in addition to those defined by ADO. As with the ADO connection properties, these provider-specific properties can be set using the **ConnectionString** property of the **Connection** object.

Parameter	Description
Trusted_Connection	Indicates the user authentication mode. This can be
	set to Yes or No. The default value is No. If this
	property is set to Yes, SQLOLEDB uses Microsoft
	Windows Authentication (recommended) to
	authorize user access to the Microsoft® SQL
	Server TM database specified by the Location and
	Datasource property values. If this property is set to
	No, SQLOLEDB uses Mixed Mode to authorize user
	access to the SQL Server database. The SQL Server
	login and password are specified in the User ID and
	Password properties.

Current Language	Indicates a SQL Server language name. Identifies the
Current Lunguage	language used for system message selection and
	formatting. The language must be installed on the
	computer running SQL Server; otherwise, opening
	the connection will fail.
Network Address	
Network Address	Indicates the network address of the instance of SQL Server specified by the Location property.
Network Library	Indicates the name of the network library (DLL) used
	to communicate with the SQL Server. The name
	should not include the path or the .dll file name
	extension. The default is provided by the SQL Server
	client configuration.
Use Procedure for	Determines whether SQL Server creates temporary
Prepare	stored procedures when commands are prepared
	using the Prepared property.
Auto Translate	Indicates whether OEM/ANSI characters are
	converted. This property can be set to True or False.
	The default value is True. If this property is set to
	True, SQLOLEDB performs OEM/ANSI character
	conversion when multibyte character strings are
	retrieved from, or sent to, SQL Server. If this
	property is set to False, SQLOLEDB does not
	perform OEM/ANSI character conversion on
	multibyte character string data.
Packet Size	Indicates a network packet size in bytes. The packet
	size property value must be from 512 through 32767.
	The default SQLOLEDB network packet size is
	4096.
Application Name	Indicates the client application name.
Workstation ID	A string identifying the workstation.

See Also

Connecting to a SQL Server Data Source

Using the Connection Object

Record Object

A **Record** object represents one row of data, and has some conceptual similarities with a one-row **Recordset**. An application can retrieve **Record** objects directly from the provider instead of a one-row **Recordset**, for example when an SQL query that selects only one row (singleton select) is executed. It is much more efficient for an application to use a **Record** object than a **Recordset** object if only one row is to be retrieved from a query.

A **Record** object also can be obtained directly from a **Recordset** object.

Recordset Object

You use **Recordset** objects to manipulate data from the provider. When you use ADO, you manipulate data almost entirely using **Recordset** objects. All **Recordset** objects consist of records (rows) and fields (columns).

In addition to the standard ADO properties, these dynamic properties are added to the **Properties** collection of the **Recordset** object.

Dynamic Properties

Access Order Property	Notification Granularity Property
Blocking Storage Objects Property	Notification Phases Property
Bookmark Type Property	Objects Transacted Property
Bookmarkable Property	Others' Changes Visible Property
Change Inserted Rows Property	Others' Inserts Visible Property
Column Privileges Property	Own Changes Visible Property
Column Set Notification Property	Own Inserts Visible Property
Command Time Out Property	Preserve on Abort Property
Defer Column Property	Preserve on Commit Property
Delay Storage Object Updates	Quick Restart Property
Property	
Fetch Backwards Property	Reentrant Events Property
Hold Rows Property	Remove Deleted Rows Property
IAccessor Property	Report Multiple Changes Property
IColumnsInfo Property	Return Pending Inserts Property
IColumnsRowset Property	Row Delete Notification Property
IConnectionPointContainer Property	Row First Change Notification
	Property
IConvertType Property	Row Insert Notification Property
Immobile Rows Property	Row Privileges Property
IRowset Property	Row Resynchronization Notification
	Property
IRowsetChange Property	Row Threading Model Property

IRowsetIdentity Property	Row Undo Change Notification
	Property
IRowsetInfo Property	Row Undo Delete Notification
	Property
IRowsetLocate Property	Row Undo Insert Notification
	<u>Property</u>
IRowsetResynch Property	Row Update Notification Property
IRowsetScroll Property	Rowset Fetch Position Change
	Notification Property
IRowsetUpdate Property	Rowset Release Notification
	Property
ISequentialStream Property	Scroll Backwards Property
ISupportErrorInfo Property	Server Cursor Property
Literal Bookmarks Property	Skip Deleted Bookmarks Property
Literal Row Identity Property	Strong Row Identity Property
Maximum Open Rows Property	Unique Rows Property
Maximum Pending Rows Property	Updatability Property
Maximum Rows Property	Use Bookmarks Property

See Also

Using the Recordset Object

Stream Object

In tree-structured hierarchies such as a file system or an e-mail system, a **Record** object may have a default binary stream of bits associated with it that contains the contents of the file or the e-mail. A **Stream** object can be used to manipulate fields or records containing these streams of data. A **Stream** object can be obtained:

- From a URL pointing to an object (typically a file) containing binary or text data. This object can be a simple document, a **Record** object representing a structured document, or a folder.
- By opening the default **Stream** object associated with a **Record** object. You can obtain the default stream associated with a **Record** object when the **Record** is opened, to eliminate a round-trip just to open the stream.
- By instantiating a **Stream** object. These **Stream** objects can be used to store data for the purposes of your application. Unlike a **Stream** associated with a URL, or the default **Stream** of a **Record**, an instantiated **Stream** has no association with an underlying source by default.

Dynamic Properties

The Microsoft OLE DB Provider for SQL Server (SQLOLEDB) inserts a number of dynamic properties into the **Properties** collection of the unopened **Connection**, **Recordset**, and **Command** objects.

Each of the topics in this section cross-reference a dynamic property with the ADO objects to which it applies, and the corresponding OLE DB property to which it maps. ADO dynamic properties either map to standard OLE DB properties, or to provider-specific OLE DB properties.

Provider-Specific Dynamic Properties

Properties in the form of DBPROP_PROPERTYNAME are standard OLE DB properties. Properties in the form of SSPROP_PROPERTYNAME are provider-specific OLE DB properties. SQLOLEDB supports these provider-specific dynamic properties:

Property Name	Description
Base Path Property	The Base Path property specifies a file
	path or URL to use for resolving relative
	paths in a template (for example, XSL on a
	template root directory, sql:mapping-
	schema attribute on a sql:xpath:query,
	external schema references in an inline
	schema, or Mapping Schema and XML
	Root properties).
<u>Content Type Property</u>	The Content Type property returns the
	output content type of an XML
	transmission.
Cursor Auto Fetch Property	The Cursor Auto Fetch property specifies
	whether the initial result set is returned
	when a cursor is opened.
Defer Prepare Property	The Defer Prepare property specifies
	whether to prepare and perform the initial
	execution of a statement in a single

	operation.
Mapping Schema Property	The Mapping Schema property specifies a
	file name or URL that points to the
	mapping schema used by the provider to
	translate an XPath command.
SS STREAM FLAGS Property	The SS STREAM FLAGS property
	specifies how an application manages
	mapping schemas, XSL files, and
	templates.
XML Root Property	The XML Root property provides a root
	tag in which the query result is wrapped to
	return a well-formed document.
XSL Property	The XSL property specifies an XSL file
	name or URL applied to the result of a
	query.

Examples

Applications reference ADO dynamic properties though the ADO **Properties** collection using this VBScript syntax:

Dim adoCmd
Set adoCmd = CreateObject("ADODB.Command")

```
Dim sBasePath
sBasePath = adoCmd.Properties("Base Path")
'Or
adoCmd.Properties("Base Path") = "C:\Inetpub\wwwroot\myvroot\"
```

A

Access Order Property

The **Access Order** property sets the order in which columns must be accessed by methods that operate on recordsets, rows, and streams.

Applies To

Command Object

Recordset Object

Syntax

object.Properties("Access Order") [= value]

Parts

object

Expression that evaluates to an object in the Applies To list.

value

Integer specifying the access order as described in Settings.

Settings

escription
Columns can be
ccessed in any
rder.
Columns bound
s storage
bjects can be
ccessed only in
equential order
s determined by
ne column
c r c s b c e s

	ordinal. Storage objects from one row must be retrieved before retrieving any columns in any subsequent row.
DBPROPVAL_AO_SEQUENTIAL	All columns must be accessed in sequential order determined by the column ordinal. Further, all columns from one row must be retrieved before retrieving any columns in any subsequent row.

Data Type

adInteger

Modifiable

Read/write

OLE DB Property

DBPROP_ACCESSORDER

Remarks

For optimal performance and interoperability, applications should set the **Access Order** property to DBPROPVAL_AO_SEQUENTIALSTORAGEOBJECTS.

Active Sessions Property

The **Active Sessions** property returns the maximum number of session objects that can be active at one time.

Applies To

Connection Object

Syntax

object.Properties("Active Sessions")

Parts

object

Expression that evaluates to an object in the Applies To list.

Data Type

adInteger

Modifiable

Read-only

OLE DB Property

DBPROP_ACTIVESESSIONS

Asynchable Abort Property

The **Asynchable Abort** property indicates whether transactions can be aborted asynchronously.

Applies To

Connection Object

Syntax

object.Properties("Asynchable Abort")

Parts

object

Expression that evaluates to an object in the Applies To list.

Returns

One of these settings.

Value	Description
	Transactions can be aborted asynchronously.
	Transactions cannot be aborted asynchronously.

Data Type

adBoolean

Modifiable

Read-only

OLE DB Property

DBPROP_ASYNCTXNABORT

See Also

Asynchable Commit Property

Asynchable Commit Property

The **Asynchable Commit** property indicates whether transactions can be committed asynchronously.

Applies To

Connection Object

Syntax

object.Properties("Asynchable Commit")

Parts

object

Expression that evaluates to an object in the Applies To list.

Returns

One of these settings.

Value	Description
True	Transactions can be committed asynchronously.
False	Transactions cannot be committed asynchronously.

Data Type

adBoolean

Modifiable

Read-only

OLE DB Property

DBPROP_ASYNCTXNCOMMIT

See Also

Asynchable Abort Property

Autocommit Isolation Levels Property

The **Autocommit Isolation Levels** property specifies the transaction isolation levels while in auto-commit mode.

Applies To

Connection Object

Syntax

object.Properties("Autocommit Isolation Levels") [= value]

Parts

object

Expression that evaluates to an object in the Applies To list.

value

Integer specifying supported transaction isolation levels as described in Settings.

Settings

A bitmask consisting of zero or more of these settings.

Constant	Description
DBPROPVAL_TI_BROWSE	Equivalent to
	DBPROPVAL_TI_READUNCOM
DBPROPVAL_TI_CURSORSTABILITY	Equivalent to
	DBPROPVAL_TI_READCOMMI
DBPROPVAL_TI_ISOLATED	Equivalent to
	DBPROPVAL_TI_SERIALIZABL
DBPROPVAL_TI_READCOMMITTED	A transaction operating at the Read
	Committed level cannot see change

	by other transactions until those tran are committed. At this level of isola dirty reads are not possible, but nonrepeatable reads and phantoms a possible.
DBPROPVAL_TI_READUNCOMMITTED	Uncommitted level can see uncomn changes made by other transactions level of isolation, dirty reads, nonre
DBPROPVAL_TI_REPEATABLEREAD	reads, and phantoms are all possible A transaction operating at the Repea Read level is guaranteed not to see a changes made by other transactions values it has already read. At this le isolation, dirty reads and nonrepeata reads are not possible, but phantoms possible.
DBPROPVAL_TI_SERIALIZABLE	A transaction operating at the Serial level guarantees that all concurrent transactions interact only in ways th produce the same effect as though e transaction were entirely executed c the other. At this isolation level, dir nonrepeatable reads, and phantoms possible.

Data Type

adInteger

Modifiable

Read/write

OLE DB Property

DBPROP_SESS_AUTOCOMMITISOLEVELS

B

Base Path Property

The **Base Path** property specifies a file path or URL to use for resolving relative paths in a template (for example, XSL on a template root directory, sql:mapping-schema attribute on a sql:xpath:query, external schema references in an inline schema, or **Mapping Schema** and **XML Root** properties).

Applies To

Command Object

Syntax

object.Properties("Base Path") [= value]

Parts

object

Expression that evaluates to an object in the Applies To list.

value

String that specifies a file name or URL.

Data Type

adBSTR

Modifiable

Read/write

OLE DB Property

SSPROP_STREAM_BASEPATH

See Also

Content Type Property

Mapping Schema Property

SS STREAM FLAGS Property

XML Root Property

XSL Property

Blocking Storage Objects Property

The **Blocking Storage** property indicates whether storage objects might prevent use of other methods on the recordset.

Applies To

Command Object

Recordset Object

Syntax

object.Properties("Blocking Storage Objects") [= value]

Parts

object

Expression that evaluates to an object in the Applies To list.

value

definition

Settings

Value	Description
True	Instantiated storage objects might prevent the use of other methods on the recordset. For example, after a storage object is created and before it is released, methods other than those on the storage object might return E_UNEXPECTED.
False	Instantiated storage objects do not prevent the use of other methods.

Data Type

adBoolean

Modifiable

Read/write

OLE DB Property

DBPROP_BLOCKINGSTORAGEOBJECTS

Bookmark Type Property

The **Bookmark Type** property indicates the bookmark type supported by the recordset.

Applies To

Command Object Recordset Object

Syntax

object.Properties("Bookmark Type") [= value]

Parts

object

Expression that evaluates to an object in the Applies To list.

value

Integer specifying the bookmark type as specified in settings.

Settings

Constant	Description
DBPROPVAL_BMK_NUMERIC	The bookmark type is numeric. Numeric
	bookmarks are based on a row property
	that is not dependent on the values in the
	columns of the row. For example, they
	can be based on the absolute position of
	the row within a recordset or on a row
	ID that the storage engine assigns to the
	row at its creation. The validity of
	numeric bookmarks is not changed by
	modifying the columns in a row.

DBPROPVAL_BMK_KEY	The bookmark type is key. Key
	bookmarks are based on the values of
	one or more of the columns in a row.
	These values form a unique key for each
	row. A key bookmark may be left
	dangling if the key values of the
	corresponding row are changed.

Data Type

adInteger

Modifiable

Read/write

OLE DB Property

DBPROP_BOOKMARKTYPE

Bookmarkable Property

The **Bookmarkable** property specifies whether a recordset supports bookmarks.

Applies To

Command Object Recordset Object

Syntax

object.Properties("Bookmarkable") [= value]

Parts

object

Expression that evaluates to an object in the Applies To list.

value

Boolean that specifies bookmark support as described in Settings.

Settings

Value	Description
True	The recordset supports the specified interface. This setting implicitly causes the created recordset to support bookmarks, setting the Use Bookmarks property to True.
False	The recordset does not support the specified interface.

Data Type

adBoolean

Modifiable

Read/write

OLE DB Property

DBProp_IRowsetLocate

See Also

Literal Bookmarks Property

Use Bookmarks Property

С

Catalog Location Property

The **Catalog Location** property indicates the position of the catalog name in a qualified table name in a text command.

Applies To

Connection Object

Syntax

object.Properties("Catalog Location")

Parts

object

Expression that evaluates to an object in the Applies To list.

Settings

Constant	Description
DBPROPVAL_CL_START	The catalog name is at the start of the fully qualified name.
DBPROPVAL_CL_END	The catalog name is at the end of the fully qualified name.

Data Type

adInteger

Modifiable

Read-only

OLE DB Property

DBPROP_CATALOGLOCATION

Catalog Term Property

The **Catalog Term** property returns the name the data source object uses for a catalog (for example, catalog, database, or directory).

Applies To

Connection Object

Syntax

object.Properties("Catalog Term")

Parts

object

Expression that evaluates to an object in the Applies To list.

Data Type

adBSTR

Modifiable

Read-only

OLE DB Property

DBPROP_CATALOGTERM

Remarks

An application can use the value returned by the **Catalog Term** property in user interfaces.

See Also

Procedure Term PropertySchema Term PropertyTable Term Property

Change Inserted Rows Property

The **Change Inserted Rows** property specifies whether an application can call the **Delete** or **Update** methods on a newly inserted row.

Applies To

Command Object Recordset Object

Syntax

object.Properties("Change Inserted Rows") [= value]

Parts

object

Expression that evaluates to an object in the Applies To list.

value

Boolean that indicates whether an application can call the **Delete** or **Update** methods as specified in Settings.

Settings

Value	Description
True	The consumer can call the Delete or
	Update methods on newly inserted rows.
False	If the consumer calls the Delete or Update
	method for newly inserted rows, Delete
	returns a status of
	DBROWSTATUS_E_NEWLYINSERTED
	for the row and Update returns
	DB_E_NEWLYINSERTED.

Data Type

adBoolean

Modifiable

Read/write

OLE DB Property

DBPROP_CHANGEINSERTEDROWS

Remarks

A newly inserted row is defined to be a row for which the insertion has been transmitted to the data source, as opposed to a pending insert row.

Column Definition Property

The **Column Definition** property returns valid clauses that can be used in column definition.

Applies To

Connection Object

Syntax

object.Properties("Column Definition")

Parts

object

Expression that evaluates to an object in the Applies To list.

Data Type

adInteger

Modifiable

Read-only

OLE DB Property

DBPROP_COLUMNDEFINITION

Remarks

Column Definition returns a bitmask defining the valid clauses for the definition of a column. For example, if **Column Definition** returns DBPROPVAL_CD_NOTNULL, the NOT NULL clause is supported.

Column Privileges Property

The **Column Privileges** property indicates whether access rights are restricted on a column-by-column basis.

Applies To

Command Object Reco

Recordset Object

Syntax

object.Properties("Column Privileges")

Parts

object

Expression that evaluates to an object in the Applies To list.

Returns

One of the following settings:

Value	Description
True	Access rights are restricted on a column-by-column basis. The
	Update method cannot be called in a query that would specify a
	column for which the user has no read access rights.
False	Access rights are not restricted on a column-by-column basis.
	The Update method can be called for any column in the
	recordset.

Data Type

adBoolean

Modifiable

Read-only

OLE DB Property

DBPROP_COLUMNRESTRICT

Remarks

If access is restricted both by row and by column, individual columns of particular rows might have their own stricter access rights, therefore the application might not even be permitted to read such columns. In this case, the column values are returned as NULL. If schema rules prevent a NULL value, the recordset should not count or return any rows that would have this condition.

See Also

Row Privileges Property

Column Set Notification Property

The **Column Set Notification** property specifies whether the notification phase is cancelable.

Applies To

Command Object Recordset Object

Syntax

object.Properties("Column Set Notification")

Parts

object

Expression that evaluates to an object in the Applies To list.

Returns

A bitmask containing zero or more of these settings.

- DBPROPVAL_NP_OKTODO
- DBPROPVAL_NP_ABOUTTODO
- DBPROPVAL_NP_SYNCHAFTER

Data Type

adInteger

Modifiable

Read-only

OLE DB Property

DBPROP_NOTIFYCOLUMNSET

Command Time Out Property

The **Command Time Out** property specifies the number of seconds before a command times out.

Applies To

Recordset Object

Syntax

object.Properties("Command Time Out") [= value]

Parts

object

Expression that evaluates to an object in the Applies To list.

value

Integer that specifies the number of seconds before a command times out.

Data Type

adInteger

Modifiable

Read/write

OLE DB Property

DBPROP_COMMANDTIMEOUT

Remarks

The **Command Time Out** property applies to any commands sent to the database. A value of zero indicates an infinite time-out.

Connect Timeout Property

The **Connect Timeout** property specifies the amount of time in seconds to wait for connection initialization to complete.

Applies To

Connection Object

Syntax

object.Properties("Connect Timeout") [= value]

Parts

object

Expression that evaluates to an object in the Applies To list.

value

Integer that specifies the number of seconds to wait for connection initialization to complete.

Data Type

adInteger

Modifiable

Read/write

OLE DB Property

DBPROP_INIT_TIMEOUT

Content Type Property

The **Content Type** property returns the output content type of an XML transmission.

Applies To

Command Object

Syntax

object.Properties("Content Type")

Parts

object

Expression that evaluates to an object in the Applies To list.

Data Type

adBSTR

Modifiable

Read-only

OLE DB Property

SSPROP_STREAM_CONTENTTYPE

Returns

Value	Description
image/jpeg	Indicates that image is the general type
	and JPEG is the specific format of the

	data.
text/html	Indicates that text is the general type and HTML is the specific format of
	the data.
text/XML	Default. Indicates that text is the general type and XML is the specific format of the data.

Remarks

The value returned by the **Content Type** property describes the data contained in the body fully enough that the receiving agent, or Web browser, can pick an appropriate mechanism to present the data to the user.

Content Type provides the content-type and the subtype, which describes the nature of the data. The content-type (such as text, image, audio, video, and so on) describes the general type of data, and the subtype specifies a specific format for that type of data.

The value returned by **Content Type** becomes the **content-type** field that is sent to the browser as part of the HTTP header, which contains the MIME-type (Multipurpose Internet Mail Extensions) of the document being sent as the body.

When a query specifies a Microsoft® SQL Server[™] BLOB field (for example, a JPEG image), **Content Type** returns **image/jpeg**. Many BLOB types have corresponding MIME types. A full list of registered MIME types is maintained by IANA (Internet Assigned Numbers Authority).

See Also

Base Path Property Mapping Schema Property SS STREAM FLAGS Property URL Access XML Root Property XSL Property

Current Catalog Property

The **Current Catalog** property specifies the name of the current catalog.

Applies To

Connection Object

Syntax

object.Properties("Current Catalog") [= value]

Parts

object

Expression that evaluates to an object in the Applies To list.

value

String that specifies the name of the current catalog.

Data Type

adBSTR

Modifiable

Read/write

OLE DB Property

DBPROP_CURRENTCATALOG

Cursor Auto Fetch Property

The **Cursor Auto Fetch** property specifies whether the initial result set is returned when a cursor is opened.

Applies To

Command Object

Syntax

object.Properties("Cursor Auto Fetch") [= value]

Parts

object

Expression that evaluates to an object in the Applies To list.

value

Boolean that indicates when the initial result set is returned as specified in Settings.

Settings

Value	Description
True	When a cursor is opened, the initial result set is returned.
False	Default. When a cursor is opened, no results are returned.

Data Type

adBoolean

Modifiable

Read/write

OLE DB Property

SSPROP_CURSORAUTOFETCH

Remarks

Setting **Cursor Auto Open** to True can result in a performance enhancement by avoiding an extra round trip to the server.

D

Data Source Property

The **Data Source** property specifies the name of the database to which to connect.

Applies To

Connection Object

Syntax

object.Properties("Data Source") [= value]

Parts

object

Expression that evaluates to an object in the Applies To list.

value

String that specifies the name of the database.

Data Type

adBSTR

Modifiable

Read/write

OLE DB Property

DBPROP_INIT_DATASOURCE

Remarks

An ADO application can also use the OLE DB **Server** keyword, which is equivalent to the **Data Source** property.

Data Source Name Property

The **Data Source Name** property returns the name of the data source object, and is typically used during the connection process.

Applies To

Connection Object

Syntax

object.Properties("Data Source Name")

Parts

object

Expression that evaluates to an object in the Applies To list.

Data Type

adBSTR

Modifiable

Read-only

OLE DB Property

DBPROP_DATASOURCENAME

Data Source Object Threading Model Property

The **Data Source Object Threading Model** property specifies which threading models are supported by the data source.

Applies To

Connection Object

Syntax

object.Properties("Data Source Object Threading Model")

Parts

object

Expression that evaluates to an object in the Applies To list.

Settings

A bitmask containing one or more of these settings:

Constant	Description
DBPROPVAL_RT_APTMTTHREAD	The apartment thread model is
	supported.
DBPROPVAL_RT_FREETHREAD	The free thread model is supported.
DBPROPVAL_RT_SINGLETHREAD	The single thread model is
	supported.

Data Type

adInteger

Modifiable

Read-only

OLE DB Property

DBPROP_DSOTHREADMODEL

Remarks

In the case where additional threading limitations are imposed by underlying components, the threading model returned by the **Data Source Object Threading Model** property might be stricter than the thread model registered under the CLSID of the provider.

DBMS Name Property

The **DBMS Name** property returns the name of the product accessed by the provider (for example, Microsoft® SQL Server[™], or Microsoft Excel).

Applies To

Connection Object

Syntax

object.Properties("DBMS Name")

Parts

object

Expression that evaluates to an object in the Applies To list.

Data Type

adBSTR

Modifiable

Read-only

OLE DB Property

DBPROP_DBMSNAME

See Also

DBMS Version Property

DBMS Version Property

The **DBMS Version** property returns the version of the product accessed by the provider.

Applies To

Connection Object

Syntax

object.Properties("DBMS Version")

Parts

object

Expression that evaluates to an object in the Applies To list.

Data Type

adBSTR

Modifiable

Read-only

OLE DB Property

DBPROP_DBMSVER

Remarks

The version is of the form ##.##.####, where the first two digits are the major version, the next two digits are the minor version, and the last four digits are the release version. The provider must render the product version in this form but

can also append the product-specific version (for example, 04.01.0000 Rdb 4.1).

See Also

DBMS Name Property

OLE DB Version Property

Defer Column Property

The **Defer Column** property specifies when data in a column is fetched.

Applies To

Command Object Recordset Object

Syntax

object.Properties("Defer Column") [= value]

Parts

object

Expression that evaluates to an object in the Applies To list.

value

Boolean that indicates when data is fetched, as specified in Settings.

Settings

Value	Description
	The data in the column is not fetched until an accessor is used on the column.
	The data in the column is fetched when the row containing it is fetched.

Data Type

adBoolean

Modifiable

Read/write

OLE DB Property

DBPROP_DEFERRED

Defer Prepare Property

The **Defer Prepare** property specifies whether to prepare and perform the initial execution of a statement in a single operation.

Applies To

Command Object

Syntax

object.Properties("Defer Prepare") [= value]

Parts

object

Expression that evaluates to an object in the Applies To list.

value

Boolean that indicates how to prepare and perform initial execution of a statement, as specified in Settings.

Settings

Value	Description
	A statement is prepared and initially executed in a single operation.
	Default. A statement is prepared, and then executed in two operations.

Data Type

adBoolean

Modifiable

Read/write

OLE DB Property

SSPROP_DEFERPREPARE

Remarks

Setting **Defer Prepare** to True can result in a performance enhancement by avoiding an extra round trip to the server when a statement must be executed repeatedly.

Note If an application calls **Defer Prepare** on an instance of Microsoft® SQL Server[™] version 7.0, the operation is ignored.

Delay Storage Object Updates Property

The **Delay Storage Object Updates** property specifies whether changes to storage objects are immediately transmitted to the data source when delayed update mode is in effect.

Applies To

Command Object Revealed Reve

Recordset Object

Syntax

object.Properties("Delay Storage Object Updates") [= value]

Parts

object

Expression that evaluates to an object in the Applies To list.

value

Boolean that indicates whether changes to storage objects are immediately transmitted, as specified in Settings.

Settings

Value	Description
True	Changes to the object are not transmitted to the data source until the Update method is called. CancelBatch undoes any pending changes.
False	Changes to the object are immediately transmitted to the data source object. The Update method has no effect on the object. CancelBatch does not undo changes made to the object since the row was last fetched or updated.

Data Type

adBoolean

Modifiable

Read/write

OLE DB Property

DBPROP_DELAYSTORAGEOBJECTS

Remarks

The **Delay Storage Object Updates** property has no effect on storage objects in immediate update mode.

Ε

Extended Properties Property

The **Extended Properties** property sets or retrieves provider-specific connection information that cannot be explicitly described through the property mechanism.

Applies To

Connection Object

Syntax

object.Properties("Extended Properties") [= value]

Parts

object

Expression that evaluates to an object in the Applies To list.

value

String containing provider-specific, extended connection information.

Data Type

adBSTR

Modifiable

Read/write

OLE DB Property

DBPROP_INIT_PROVIDERSTRING

Remarks

Use of this property implies that the application developer knows how this string will be interpreted and used by the provider. Applications should use this property only for provider-specific connection information that cannot be explicitly described through the property mechanism.

 \mathbf{F}

Fetch Backwards Property

The **Fetch Backwards** property indicates whether the recordset can fetch backward.

Applies To

Command Object Recordset Object	
---------------------------------	--

Syntax

object.Properties("Fetch Backwards") [= value]

Parts

object

Expression that evaluates to an object in the Applies To list.

value

Boolean that indicates whether the recordset can fetch backward, as specified in Settings.

Settings

Value	Description
True	The recordset can fetch backward.
False	The recordset cannot fetch backward.

Data Type

adBoolean

Modifiable

Read/write

OLE DB Property

DBPROP_CANFETCHBACKWARDS

G

GROUP BY Support Property

The **GROUP BY Support** property indicates the relationship between the columns in a GROUP BY clause and the nonaggregated columns in a SELECT statement.

Applies To

Connection Object

Syntax

object.Properties("GROUP BY Support")

Parts

object

Expression that evaluates to an object in the Applies To list.

Returns

One of the following settings:

Constant	Description
DBPROPVAL_GB_EQUALS_SELECT	The GROUP BY clause must
	contain all nonaggregated
	columns in the select list. It
	cannot contain any other
	columns (for example,
	SELECT DEPT,
	MAX(SALARY) FROM
	EMPLOYEE GROUP BY
	DEPT).
DBPROPVAL_GB_COLLATE	A COLLATE clause can be

	specified at the end of each
	grouping column.
DBPROPVAL_GB_CONTAINS_SELECT	The GROUP BY clause must
	contain all nonaggregated
	columns in the select list. It can
	contain columns that are not in
	the select list (for example,
	SELECT DEPT,
	MAX(SALARY) FROM
	EMPLOYEE GROUP BY
	DEPT, AGE).
DBPROPVAL_GB_NO_RELATION	The columns in the GROUP
	BY clause and the SELECT
	statement are not related (for
	example, SELECT DEPT,
	SALARY FROM EMPLOYEE
	GROUP BY DEPT, AGE).

Data Type

adInteger

Modifiable

Read-only

OLE DB Property

DBPROP_GROUPBY

\mathbf{H}

Heterogeneous Table Support Property

The **Heterogeneous Table Support** property indicates whether the provider can join tables from different catalogs or providers.

Applies To

Connection Object

Syntax

object.Properties("Heterogeneous Table Support")

Parts

object

Expression that evaluates to an object in the Applies To list.

Returns

A bitmask containing zero or more of these settings.

Constant	Description
DBPROPVAL_HT_DIFFERENT_CATALOGS	Tables from different
	catalogs can be joined.
DBPROPVAL_HT_DIFFERENT_PROVIDERS	Tables from different
	providers can be joined.

Data Type

adInteger

Modifiable

Read-only

OLE DB Property

DBPROP_HETEROGENEOUSTABLES

Hold Rows Property

The **Hold Rows** property specifies whether the recordset allows the application to retrieve more rows or change the next fetch position, while holding previously fetched rows or rows with pending changes.

Applies To

Command Object Recordset Object

Syntax

object.Properties("Hold Rows") [= value]

Parts

object

Expression that evaluates to an object in the Applies To list.

value

Boolean specifying whether the application can retrieve more rows, as described in Settings.

Settings

Value	Description
True	The recordset allows the application to retrieve more rows or change the next fetch position, while holding previously fetched rows or rows with pending changes.
False	The recordset requires pending changes to be transmitted to the database and all rows to be released before fetching additional
	rows, inserting new rows, or changing the next fetch position.

Data Type

adBoolean

Modifiable

Read/write

OLE DB Property

DBPROP_CANHOLDROWS

Remarks

If the provider makes no optimizations for releasing all rows between fetches, it is not required to return DB_E_ROWSNOTRELEASED when retrieving rows without releasing the previously held set of row handles. Such providers do not return an error when setting the **Hold Rows** property to False, but always return True if the consumer calls **Hold Rows**.

I

IAccessor Property

The **IAccessor** property indicates whether the provider supports the OLE DB **IAccessor** interface.

Applies To

Command Object Recordset Object	Command Object	Recordset Object
---------------------------------	----------------	------------------

Syntax

object.Properties("IAccessor")

Parts

object

Expression that evaluates to an object in the Applies To list.

Data Type

adBoolean

Modifiable

Read-only

OLE DB Property

DBPROP_IAccessor

Remarks

ADO requires that the provider support the OLE DB **IAccessor** interface. The value of the **IAccessor** property is read-only and is always set to True, indicating that the recordset supports the specified interface. The value of this property

cannot be set to False.

The **IAccessor** property is useful for conformance testing.

IColumnsInfo Property

The **IColumnsInfo** property indicates whether the provider supports the OLE DB **IColumnsInfo** interface.

Applies To

Command Object Recordset Object

Syntax

object.Properties("IColumnsInfo") [= value]

Parts

object

Expression that evaluates to an object in the Applies To list.

Data Type

adBoolean

Modifiable

Read-only

OLE DB Property

DBPROP_IColumnsInfo

Remarks

ADO requires that the provider support the OLE DB **IColumnsInfo** interface. The value of the **IColumnsInfo** property is read-only and is always set to True, indicating that the recordset supports the specified interface. The value of this property cannot be set to False.

The **IColumnsInfo** property is useful for conformance testing.

IColumnsRowset Property

The **IColumnsRowset** property indicates whether the provider supports the OLE DB **IColumnsRowset** interface.

Applies To

Command Object Recordset Object

Syntax

object.Properties("IColumnsRowset") [= value]

Parts

object

Expression that evaluates to an object in the Applies To list.

value

Boolean that indicates whether the provider supports the specified interface, as specified in Settings.

Settings

Value	Description
True	The recordset supports the specified interface.
False	The recordset does not support the specified interface.

Data Type

adBoolean

Read/write

OLE DB Property

DBPROP_IColumnsRowset

Remarks

ADO uses the OLE DB **IColumnsRowset** interface if the provider supports it.

The **IColumnsRowset** property is useful for conformance testing.

IConnectionPointContainer Property

The **IConnectionPointContainer** property indicates whether the provider supports the OLE DB **IConnectionPointContainer** interface.

Applies To

Command Object Recordset Object

Syntax

object.Properties("IConnectionPointContainer") [= value]

Parts

object

Expression that evaluates to an object in the Applies To list.

value

Boolean that indicates whether the provider supports the specified interface, as specified in Settings.

Value	Description
True	The recordset supports the specified interface.
False	The recordset does not support the specified interface.

Data Type

adBoolean

Modifiable

Read/write

OLE DB Property

DBPROP_IConnectionPointContainer

Remarks

ADO uses the OLE DB **IConnectionPointContainer** interface if the provider supports it.

The **IConnectionPointContainer** property is useful for conformance testing.

IConvertType Property

The **IconvertType** property indicates whether the provider supports the OLE DB **IConvertType** interface.

Applies To

Syntax

object.Properties("IConvertType")

Parts

object

Expression that evaluates to an object in the Applies To list.

Data Type

adBoolean

Modifiable

Read-only

OLE DB Property

DBPROP_IConvertType

Remarks

ADO requires that the provider support the OLE DB **IConvertType** interface. The value of the **IConvertType** property is read-only and is always set to True, indicating that the recordset supports the specified interface. The value of this property cannot be set to False.

The **IConvertType** property is useful for conformance testing.

Identifier Case Sensitivity Property

The **Identifier Case Sensitivity** property indicates how identifiers treat case in data definition commands or interfaces.

Applies To

Connection Object

Syntax

object.Properties("Identifier Case Sensitivity")

Parts

object

Expression that evaluates to an object in the Applies To list.

Returns

One of the following settings:

Constant	Description
DBPROPVAL_IC_UPPER	Identifiers in SQL are case-insensitive and
	are stored in uppercase.
DBPROPVAL_IC_LOWER	Identifiers in SQL are case-insensitive and
	are stored in lowercase.
DBPROPVAL_IC_SENSITIVE	Identifiers in SQL are case-sensitive and
	are stored in mixed case.
DBPROPVAL_IC_MIXED	Identifiers in SQL are case-insensitive and
	are stored in mixed case.

Data Type

adInteger

Modifiable

Read-only

OLE DB Property

DBPROP_IDENTIFIERCASE

Immobile Rows Property

The **Immobile Rows** property specifies whether to reorder inserted rows in a recordset.

Applies To

	Command Object	Recordset Object
--	----------------	------------------

Syntax

object.Properties("Immobile Rows") [= value]

Parts

object

Expression that evaluates to an object in the Applies To list.

value

Boolean that specifies whether to reorder inserted rows, as specified in Settings.

Settings

Value	Description
True	The recordset will not reorder inserted or updated rows. Rows appear at the end of the recordset.
False	If the recordset is ordered, inserted rows and updated rows (where one or more of the columns in the ordering criteria are updated) obey the ordering criteria of the recordset. If the recordset is not ordered, inserted rows are not guaranteed to appear in a determinate position and the position of updated rows is not changed.

Data Type

adBoolean

Modifiable

Read/write

OLE DB Property

DBPROP_IMMOBILEROWS

Remarks

This property is meaningful only if the **Own Inserts Visible** property is set to True.

See Also

Own Inserts Visible Property

Initial Catalog Property

The **Initial Catalog** property specifies the name of the initial default catalog to use when connecting to a data source.

Applies To

Connection Object

Syntax

object.Properties("Initial Catalog") [= value]

Parts

object

Expression that evaluates to an object in the Applies To list.

value

String that specifies the catalog name.

Data Type

adBSTR

Modifiable

Read/write

OLE DB Property

DBPROP_INIT_CATALOG

Remarks

An ADO application can also use the OLE DB **Database** keyword, which is equivalent to the **Initial Catalog** property.

IRowset Property

The **IRowset** property indicates whether the provider supports the OLE DB **IRowset** interface.

Applies To

Command Object	Recordset Object	
----------------	------------------	--

Syntax

object.Properties("IRowset")

Parts

object

Expression that evaluates to an object in the Applies To list.

Data Type

adBoolean

Modifiable

Read-only

OLE DB Property

DBPROP_IRowset

Remarks

ADO requires that the provider support the OLE DB **IRowset** interface. The value of the **IRowset** property is read-only and is always set to True, indicating that the recordset supports the specified interface. The value of this property

cannot be set to False.

The **IRowset** property is useful for conformance testing.

IRowsetChange Property

The **IRowsetChange** property indicates whether the provider supports the OLE DB **IRowsetChange** interface.

Applies To

Command Object Recordset Object

Syntax

object.Properties("IRowsetChange") [= value]

Parts

object

Expression that evaluates to an object in the Applies To list.

value

Boolean that indicates whether the provider supports the specified interface, as specified in Settings.

Settings

Value	Description
True	The recordset supports the specified interface.
False	The recordset does not support the specified interface.

Data Type

adBoolean

Read/write

OLE DB Property

DBPROP_IRowsetChange

Remarks

ADO uses the OLE DB **IRowsetChange** interface if the provider supports it.

The **IRowsetChange** property is useful for conformance testing.

Setting The **IRowsetUpdate** property to True automatically sets the **IRowsetChange** property to True.

When the **IRowsetChange** property is set to False, the **Updatability** property is set to zero. Any attempt to set the **Updatability** property to a value other than zero results in a conflict reported by the provider, which returns DBPROPSTATUS_CONFLICTING.

See Also

IRowsetUpdate Property

Updatability Property

IRowsetIdentity Property

The **IRowsetIdentity** property indicates whether the provider supports the OLE DB **IRowsetIdentity** interface.

Applies To

Command Object Recordset Object

Syntax

object.Properties("IRowsetIdentity") [= value]

Parts

object

Expression that evaluates to an object in the Applies To list.

value

Boolean that indicates whether the provider supports the specified interface, as specified in Settings.

Settings

Value	Description
True	The recordset supports the specified interface.
False	The recordset does not support the specified interface.

Data Type

adBoolean

Read/write

OLE DB Property

DBPROP_IRowsetIdentity

Remarks

ADO uses the OLE DB **IRowsetIdentity** interface if the provider supports it.

The **IRowsetIdentity** property is useful for conformance testing.

IRowsetInfo Property

The **IRowsetInfo** property indicates whether the provider supports the OLE DB **IRowsetInfo** interface.

Applies To

Command Object Recordset Object

Syntax

```
object.Properties("IRowsetInfo")
```

Parts

object

Expression that evaluates to an object in the Applies To list.

Data Type

adBoolean

Modifiable

Read-only

OLE DB Property

DBPROP_IRowsetInfo

Remarks

ADO requires that the provider support the OLE DB **IRowsetInfo** interface. The value of the **IRowsetInfo** property is read-only and is always set to True, indicating that the recordset supports the specified interface. The value of this

property cannot be set to False.

The **IRowsetInfo** property is useful for conformance testing.

IRowsetLocate Property

The **IRowsetLocate** property indicates whether the provider supports the OLE DB **IRowsetLocate** interface.

Applies To

Command Object Recordset Object

Syntax

object.Properties("IRowsetLocate") [= value]

Parts

object

Expression that evaluates to an object in the Applies To list.

value

Boolean that indicates whether the provider supports the specified interface, as specified in Settings.

Settings

Value	Description
True	The recordset supports the specified interface. This setting implicitly causes the created recordset to support bookmarks,
	returning True for the Use BookMarks property.
False	The recordset does not support the specified interface.

Data Type

adBoolean

Read/write

OLE DB Property

DBPROP_IRowsetLocate

Remarks

ADO uses the OLE DB **IRowsetLocate** interface if the provider supports it.

The **IRowsetLocate** property is useful for conformance testing.

IRowsetResynch Property

The **IRowsetResynch** property indicates whether the provider supports the OLE DB **IRowsetResynch** interface.

Applies To

Command Object Recordset Object

Syntax

object.Properties("IRowsetResynch") [= value]

Parts

object

Expression that evaluates to an object in the Applies To list.

value

Boolean that indicates whether the provider supports the specified interface.

Settings

Value	Description
True	The recordset supports the specified interface.
False	The recordset does not support the specified interface.

Data Type

adBoolean

Modifiable

Read/write

OLE DB Property

DBPROP_IRowsetResynch

Remarks

ADO uses the OLE DB **IRowsetResynch** interface if the provider supports it.

The **IRowsetResynch** property is useful for conformance testing.

IRowsetScroll Property

The **IRowsetScroll** property indicates whether the provider supports the OLE DB **IRowsetScroll** interface.

Applies To

Command Object Recordset Object

Syntax

object.Properties("IRowsetScroll") [= value]

Parts

object

Expression that evaluates to an object in the Applies To list.

value

Boolean that indicates whether the provider supports the specified interface, as specified in Settings.

Settings

Value	Description
True	The recordset supports the specified interface.
False	The recordset does not support the specified interface.

Data Type

adBoolean

Read/write

OLE DB Property

DBPROP_IRowsetScroll

Remarks

ADO uses the OLE DB **IRowsetScroll** interface if the provider supports it.

The **IRowsetScroll** property is useful for conformance testing.

IRowsetUpdate Property

The **IRowsetUpdate** property indicates whether the provider supports the OLE DB **IRowsetUpdate** interface.

Applies To

Command Object Recordset Object

Syntax

```
object.Properties("IRowsetUpdate") [= value]
```

Parts

object

Expression that evaluates to an object in the Applies To list.

value

Boolean that indicates whether the provider supports the specified interface, as specified in Settings.

Settings

Value	Description
True	The recordset supports the specified interface, and automatically
	sets the IRowsetChange property to True.
False	The recordset does not support the specified interface.

Data Type

adBoolean

Read/write

OLE DB Property

DBPROP_IRowsetUpdate

Remarks

ADO uses the OLE DB **IRowsetUpdate** interface if the provider supports it.

The **IRowsetUpdate** property is useful for conformance testing.

See Also

IRowsetChange Property
Updatability Property

ISequentialStream Property

The **ISequentialStream** property indicates whether the provider supports the OLE DB **ISequentialStream** interface.

Applies To

Command Object Recordset Object

Syntax

object.Properties("ISequentialStream") [= value]

Parts

object

Expression that evaluates to an object in the Applies To list.

value

Boolean that indicates whether the provider supports the specified interface, as specified in Settings.

Settings

Value	Description
True	The recordset supports the specified interface.
False	The recordset does not support the specified interface.

Data Type

adBoolean

Read/write

OLE DB Property

DBPROP_ISequentialStream

Remarks

ADO uses the OLE DB **ISequentialStream** interface if the provider supports it.

If the value of the **ISequentialStream** property is set to True, the recordset is capable of manipulating the contents of columns as a storage object supporting the specified interface. The provider reports its ability to enable this property on a per-column basis by setting the flag DBPROPFLAGS_COLUMNOK. A provider that does not have the ability to turn the property on or off on a per-column basis does not set DBPROPFLAGS_COLUMNOK.

Whether or not the property is supported in the recordset as a whole or on a percolumn basis, the ability to manipulate a column value as a storage object depends on whether the provider supports the coercion from the column's native type (BLOB or non-BLOB) to the particular storage interface.

Isolation Levels Property

The **Isolation Levels** property specifies the supported transaction isolation levels.

Applies To

Connection Object

Syntax

object.Properties("Isolation Levels")

Parts

object

Expression that evaluates to an object in the Applies To list.

Returns

A bitmask containing zero or more of these settings.

Constant	Description
DBPROPVAL_TI_BROWSE	Equivalent to
	DBPROPVAL_TI_READUNCOM
DBPROPVAL_TI_CURSORSTABILITY	Equivalent to
	DBPROPVAL_TI_READCOMMI
DBPROPVAL_TI_ISOLATED	Equivalent to
	DBPROPVAL_TI_SERIALIZABL
DBPROPVAL_TI_READCOMMITTED	A transaction operating at the Read
	Committed level cannot see change
	by other transactions until those tran
	are committed. At this level of isola
	dirty reads are not possible but noni
	reads and phantoms are possible.

	Provide and Language and Language
DBPROPVAL_TI_READUNCOMMITTED	1 0
	Uncommitted level can see uncomn
	changes made by other transactions
	level of isolation, dirty reads, nonre
	reads, and phantoms are all possible
DBPROPVAL_TI_REPEATABLEREAD	A transaction operating at the Repea
	Read level is guaranteed not to see a
	changes made by other transactions
	it has already read. At this level of i
	dirty reads and nonrepeatable reads
	possible but phantoms are possible.
DBPROPVAL_TI_SERIALIZABLE	A transaction operating at the Serial
	level guarantees that all concurrent
	transactions interact only in ways th
	produce the same effect as if each ti
	were entirely executed one after the
	this isolation level, dirty reads, noni
	reads, and phantoms are not possibl

Data Type

adInteger

Modifiable

Read-only

OLE DB Property

DBPROP_SUPPORTEDTXNISOLEVELS

Isolation Retention Property

The **Isolation Retention** property specifies the supported transaction isolation retention levels.

Applies To

Connection Object

Syntax

object.Properties("Isolation Retention")

Parts

object

Expression that evaluates to an object in the Applies To list.

Returns

A bitmask containing zero or more of these settings.

Constant	Description
DBPROPVAL_TR_ABORT	The transaction preserves its isolation context across a retaining abort.
DBPROPVAL_TR_ABORT_DC	The transaction may either preserve or dispose of isolation context across a retaining abort.
DBPROPVAL_TR_ABORT_NO	The transaction is explicitly not to preserve isolation across a retaining abort.
DBPROPVAL_TR_BOTH	Isolation is preserved across both a retaining commit and a retaining abort.
DBPROPVAL_TR_COMMIT	The transaction preserves its isolation

	context (that is, it preserves its locks, if that is how isolation is implemented) across a retaining commit.
DBPROPVAL_TR_COMMIT_DC	The transaction may either preserve or dispose of isolation context across a retaining commit.
DBPROPVAL_TR_COMMIT_NO	The transaction is explicitly not to preserve isolation across a retaining commit.
DBPROPVAL_TR_DONTCARE	The transaction may preserve or dispose of isolation context across a retaining commit or abort. This is the default.
DBPROPVAL_TR_NONE	Isolation is explicitly not to be retained across either a retaining commit or a retaining abort.
DBPROPVAL_TR_OPTIMISTIC	Optimistic concurrency control is to be used. If DBPROPVAL_TR_OPTIMISTIC is specified, and then whatever isolation technology is in place (such as locking), it must be the case that other transactions' ability to make changes to the data and resources manipulated by this transaction is not in any way affected by the data read or updated by this transaction. That is, optimistic control is to be used for all data in the transaction.

Data Type

adInteger

Modifiable

Read-only

OLE DB Property

DBPROP_SUPPORTEDTXNISORETAIN

ISupportErrorInfo Property

The **ISupportErrorInfo** property indicates whether the provider supports the OLE DB **ISupportErrorInfo** interface.

Applies To

Command Object Recordset Object

Syntax

object.Properties("ISupportErrorInfo") [= value]

Parts

object

Expression that evaluates to an object in the Applies To list.

value

Boolean that indicates whether the provider supports the specified interface, as specified in Settings

Settings

Value	Description
True	The recordset supports the specified interface.
False	The recordset does not support the specified interface.

Data Type

adBoolean

Read/write

OLE DB Property

DBPROP_ISupportErrorInfo

Remarks

ADO uses the OLE DB **ISupportErrorInfo** interface if the provider supports it.

The **ISupportErrorInfo** property is useful for conformance testing.

L

Literal Bookmarks Property

The **Literal Bookmarks** property specifies whether bookmarks can be compared as a sequence of bytes.

Applies To

Command Object

Recordset Object

Syntax

object.Properties("Literal Bookmarks") [= value]

Parts

object

Expression that evaluates to an object in the Applies To list.

value

Boolean that indicates whether bookmarks can be compared as a sequence of bytes, as specified in Settings

Settings

Value	Description	
True	Bookmarks can be compared literally. That is, they can be	
	compared as a sequence of bytes.	
False	Bookmarks cannot be compared literally.	

Data Type

adBoolean

Modifiable

Read/write

OLE DB Property

DBPROP_LITERALBOOKMARKS

Remarks

Setting the value of this property to True automatically sets the value of **Use BookMarks** to True.

See Also

Bookmarkable Property

Use Bookmarks Property

Literal Row Identity Property

The **Literal Row Identity** property indicates whether an application can perform a binary comparison of two row handles to determine whether they point to the same row.

Applies To

Command Object Recordset Object

Syntax

object.Properties("Literal Row Identity")

Parts

object

Expression that evaluates to an object in the Applies To list.

Returns

One of the following settings:

Value	Description	
True	The application can perform a binary comparison of two row handles to determine whether they point to the same row	
False	 handles to determine whether they point to the same row. Multiple and concurrently held row handles can represent the same row in the underlying database. To the recordset, these generally appear as separate rows. Therefore, a change made to retrieved column value is not reflected when retrieving the row through a second row handle. 	

Data Type

adBoolean

Modifiable

Read-only

OLE DB Property

DBPROP_LITERALIDENTITY

Remarks

The **Strong Row Identity** property specifies whether the handle of a newly inserted row can be successfully compared to another handle.

See Also

Strong Row Identity Property

Locale Identifier Property

The **Locale Identifier** property specifies a preferred locale ID.

Applies To

Connection Object

Syntax

object.Properties("Locale Identifier") [= value]

Parts

object

Expression that evaluates to an object in the Applies To list.

value

Integer that specifies the locale ID.

Data Type

adInteger

Modifiable

Read/write

OLE DB Property

DBPROP_INIT_LCID

Remarks

Applications specify the LCID at initialization. This provides a method for the

server to determine the application's specified LCID in cases where it can use this information. This property does not guarantee that all text returned to the application is translated according to the LCID.

Lock Mode Property

The **Lock Mode** property specifies the level of locking performed by the recordset.

Applies To

Command Object

Syntax

object.Properties("Lock Mode") [= value]

Parts

object

Expression that evaluates to an object in the Applies To list.

value

Integer that specifies the level of locking as described in Settings.

Settings

Constant	Description
DBPROPVAL_LM_NONE	The provider is not required to lock
	rows at any time to ensure successful
	updates. Updates may fail when sent to
	the server for reasons of concurrency
	(for example, if someone else has
	updated the row).
DBPROPVAL_LM_SINGLEROW	The provider uses the minimum level
	of locking necessary to ensure that
	changes successfully written to a single
	row returned by the most recent fetch

will not fail due to a concurrency violation. Therefore, using deferred update mode will not fail due to a concurrency violation. This may mean that the provider takes a lock on the row when the Update method is first called on the row, but the provider may lock the row as early as when it is read to guarantee that operations on the row, such as updates, will succeed. The implications of DBPROPVAL_LM_SINGLEROW,
1
and the Lock Mode property in
general, are the same in both immediate and deferred update modes.

Data Type

adInteger

Modifiable

Read/write

OLE DB Property

DBPROP_LOCKMODE

Remarks

Lock mode and isolation level are closely related but distinct. An application's isolation level specifies the isolation of that application from changes made by other users to the underlying data. Lock mode defines when underlying data is locked to ensure that updates succeed. The provider may use locking to enforce higher levels of isolation, in which case a higher level of locking may occur than is required to enforce the specified lock mode. The **Lock Mode** property

specifies the minimum level of locking.

M

Mapping Schema Property

The **Mapping Schema** property specifies a file name or URL that points to the mapping schema used by the provider to translate an XPath command.

Applies To

Command Object

Syntax

object.Properties("MappingSchema") [= value]

Parts

object

Expression that evaluates to an object in the Applies To list.

value

String that specifies a file name or URL.

Data Type

adBSTR

Modifiable

Read/write

OLE DB Property

SSPROP_STREAM_MAPPINGSCHEMA

Remarks

Relative paths are resolved by the **Base Path** property. If the **Base Path** property is not set, the relative path defaults to the current directory.

Mapping Schema is ignored for SQL statements and XML template queries.

See Also

Base Path Property Content Type Property SS STREAM FLAGS Property XML Root Property XSL Property

Maximum Index Size Property

The **Maximum Index Size** property returns the maximum number of bytes allowed in the combined columns of an index. If there is no specified limit or the limit is unknown, the value is set to zero.

Applies To

Connection Object

Syntax

object.Properties("Maximum Index Size")

Parts

object

Expression that evaluates to an object in the Applies To list.

Data Type

adInteger

Modifiable

Read-only

OLE DB Property

DBPROP_MAXINDEXSIZE

Maximum Open Rows Property

The **Maximum Open Rows** property specifies the maximum number of rows that can be active at the same time.

Applies To

Command Object Recordset Object

Syntax

object.Properties("Maximum Open Rows") [= value]

Parts

object

Expression that evaluates to an object in the Applies To list.

value

Integer that specifies the maximum number of rows.

Data Type

adInteger

Modifiable

Read/write

OLE DB Property

DBPROP_MAXOPENROWS

Remarks

This limit on the maximum number of rows does not reflect resource limitations such as Random Access Memory (RAM), but does apply if the recordset implementation uses some strategy that results in a limit.

If there is no limit, the value of the **Maximum Open Rows** property is set to zero. The provider is free to support a greater number of active rows than the maximum specified by the application. In this case, the provider returns the actual maximum number of active rows instead of the value specified by the application.

Maximum Pending Rows Property

The **Maximum Pending Rows** property specifies the maximum number of rows that can have pending changes at the same time.

Applies To

Command Object

Recordset Object

Syntax

object.Properties("Maximum Pending Rows") [= value]

Parts

object

Expression that evaluates to an object in the Applies To list.

value

Integer that specifies the maximum number of rows that can have pending changes.

Data Type

adInteger

Modifiable

Read/write

OLE DB Property

DBPROP_MAXPENDINGROWS

Remarks

This limit on the maximum number of pending rows does not reflect resource limitations such as Random Access Memory (RAM), but does apply if the recordset implementation uses some strategy that results in a limit.

If there is no limit, the value of the **Maximum Pending Rows** property is set to zero. The provider is free to support a greater number of pending rows than the maximum specified by the application. In this case, the provider will return the actual maximum number of pending rows instead of the value specified by the application.

Maximum Row Size Property

The **Maximum Row Size** property returns the maximum length of a single row in a table.

Applies To

Connection Object

Syntax

object.Properties("Maximum Row Size")

Parts

object

Expression that evaluates to an object in the Applies To list.

Data Type

adInteger

Modifiable

Read-only

OLE DB Property

DBPROP_MAXROWSIZE

Remarks

If there is no specified limit or the limit is unknown, **Maximum Row Size** is set to zero.

See Also

Maximum Row Size Includes BLOB Property

Maximum Row Size Includes BLOB Property

The **Maximum Row Size Includes BLOB** property indicates that the value returned by the **Maximum Row Size** property includes all BLOB data.

Applies To

Connection Object

Syntax

object.Properties("Maximum Row Size Includes BLOB")

Parts

object

Expression that evaluates to an object in the Applies To list.

Returns

One of the following settings:

Value	Description	
	The maximum row size returned by the Maximum Row Size property includes the length of all BLOB data.	
	The maximum row size returned by the Maximum Row Size property does not include the length of all BLOB data.	

Data Type

adBoolean

Modifiable

Read-only

OLE DB Property

DBPROP_MAXROWSIZEINCLUDESBLOB

See Also

Maximum Row Size Property

Maximum Rows Property

The **Maximum Rows** property specifies the maximum number of rows that can be returned in a recordset.

Applies To

Command Object Recordset Object

Syntax

object.Properties("Maximum Rows") [= value]

Parts

object

Expression that evaluates to an object in the Applies To list.

value

Integer that specifies the maximum number of rows.

Data Type

adInteger

Modifiable

Read/write

OLE DB Property

DBPROP_MAXROWS

Remarks

If no limit is specified for the **Maximum Rows** property, the value is set to zero.

If the application attempts to fetch a greater number of rows in a recordset than specified by the **Maximum Rows** property, the recordset behaves as if the table contains, or the query returns, only the quantity specified by **MaximumRows**. The provider returns DB_S_ENDOFROWSET.

Pending deletes do not count against the recordset limit specified by the **Maximum Rows** property. The provider is not required to check the **Maximum Rows** property when inserting or deleting rows.

Maximum Tables in SELECT Property

The **Maximum Tables in SELECT** property specifies the maximum number of tables allowed in the FROM clause of a SELECT statement.

Applies To

Connection Object

Syntax

object.Properties("Maximum Tables in SELECT") [= value]

Parts

object

Expression that evaluates to an object in the Applies To list.

value

Integer that specifies the maximum number of tables.

Data Type

adInteger

Modifiable

Read/write

OLE DB Property

DBPROP_MAXTABLESINSELECT

Remarks

If there is no specified limit or the limit is unknown, the **Maximum Tables in SELECT** property is set to zero.

Multiple Parameter Sets Property

The **Multiple Parameter Sets** property indicates whether a provider supports multiple parameter sets.

Applies To

Connection Object

Syntax

object.Properties("Multiple Parameter Sets")

Parts

object

Expression that evaluates to an object in the Applies To list.

Returns

One of the following settings:

Value	Description	
True	The provider supports multiple parameter sets.	
	The provider supports only a single set of parameters per execution.	

Data Type

adBoolean

Modifiable

Read-only

OLE DB Property

DBPROP_MULTIPLEPARAMSETS

Multiple Results Property

The **Multiple Results** property specifies whether the provider supports multiple results objects and what restrictions it places on these objects.

Applies To

Connection Object

Syntax

object.Properties("Multiple Results")

Parts

object

Expression that evaluates to an object in the Applies To list.

Returns

A bitmask containing zero or more of the following settings:

Constant	Description
DBPROPVAL_MR_CONCURRENT	More than one recordset created
	by the same multiple results object
	can exist concurrently. If this bit is
	not set, the consumer must release
	the current recordset before
	retrieving the next result.
DBPROPVAL_MR_NOTSUPPORTED	The provider does not support
	multiple results objects.
DBPROPVAL_MR_SUPPORTED	The provider supports multiple
	results objects.

Data Type

adInteger

Modifiable

Read-only

OLE DB Property

DBPROP_MULTIPLERESULTS

Multiple Storage Objects Property

The **Multiple Storage Objects** property indicates whether the provider supports multiple open storage objects at the same time.

Applies To

Connection Object

Syntax

object.Properties("Multiple Storage Objects")

Parts

object

Expression that evaluates to an object in the Applies To list.

Returns

One of the following settings:

Value	Description
True	The provider supports multiple, open storage objects at the same time.
	The provider supports only one open storage object at a time. Any method that attempts to open a second storage object returns a status of DBSTATUS_E_CANTCREATE for the column on which it attempted to open the second storage object, whether or not the objects are constructed over the same column, different columns in the same row, or different rows.

Data Type

adBoolean

Modifiable

Read-only

OLE DB Property

DBPROP_MULTIPLESTORAGEOBJECTS

Multi-Table Update Property

The **Multi-Table Update** property indicates whether the provider can update recordsets derived from multiple tables.

Applies To

Connection Object

Syntax

object.Properties("Multi-Table Update")

Parts

object

Expression that evaluates to an object in the Applies To list.

Returns

One of the following settings:

Value	Description	
True	The provider can update recordsets derived from multiple tables.	
False	The provider cannot update recordsets derived from multiple tables.	

Data Type

adBoolean

Modifiable

Read-only

OLE DB Property

DBPROP_MULTITABLEUPDATE

Ν

Notification Granularity Property

The **Notification Granularity** property specifies how to process modifications on multiple rows.

Applies To

$C_{1} = 1 O_{1} + 1 O_{2}$		
Command Object	Recordset Object	
<u>Commune Object</u>	<u>Itteeoraset object</u>	

Syntax

object.Properties("Notification Granularity") [= value]

Parts

object

Expression that evaluates to an object in the Applies To list.

value

Integer specifying how to process modifications, as defined in Settings.

Settings

Constant	Description
DBPROPVAL_NT_SINGLEROW	For methods that operate on multiple
	rows, the provider processes
	modifications separately for each
	phase for each row. A cancellation
	affects a single row; it does not
	affect the other rows, and
	notifications are still sent for these
	rows.
DBPROPVAL_NT_MULTIPLEROWS	For methods that operate on multiple
	rows and then for each phase, the

provider processes modifications once for all rows that succeed and once for all rows that fail. This separation can occur at each phase where a change can fail. For example, if the process deletes some rows and fails to delete others during the preliminary work phase, it processes modifications twice: once with DBEVENTPHASE_SYNCHAFTER and the array of handles of rows that it deleted, and once with
it deleted, and once with
DBEVENTPHASE_FAILEDTODO
and the array of handles of rows it
failed to delete.

adInteger

Modifiable

Read/write

OLE DB Property

DBPROP_NOTIFICATIONGRANULARITY

Remarks

The **Notification Granularity** property does not affect how providers return notifications about events that affect columns or the entire recordset.

Notification Phases Property

The **Notification Phases** property returns a bitmask specifying the notification phases supported by the provider.

Applies To

Command Object Recordset Object

Syntax

object.Properties("Notification Phases")

Parts

object

Expression that evaluates to an object in the Applies To list.

Returns

A combination of two or more of these settings.

- DBPROPVAL_NP_ABOUTTODO
- DBPROPVAL_NP_DIDEVENT
- DBPROPVAL_NP_FAILEDTODO
- DBPROPVAL_NP_OKTODO
- DBPROPVAL_NP_SYNCHAFTER

Data Type

adInteger

Modifiable

Read-only

OLE DB Property

DBPROP_NOTIFICATIONPHASES

Remarks

DBPROPVAL_NP_FAILEDTODO and DBPROPVAL_NP_DIDEVENT are returned by all providers that support notifications.

NULL Collation Order Property

The **NULL Collation Order** property indicates how NULLs are sorted in a list.

Applies To

Connection Object

Syntax

object.Properties("NULL Collation Order")

Parts

object

Expression that evaluates to an object in the Applies To list.

Returns

One of the following settings:

Constant	Description
DBPROPVAL_NC_END	NULLs are sorted at the end of the list,
	regardless of the sort order.
DBPROPVAL_NC_HIGH	NULLs are sorted at the high end of the list.
DBPROPVAL_NC_LOW	NULLs are sorted at the low end of the list.
DBPROPVAL_NC_START	NULLs are sorted at the start of the list,
	regardless of the sort order.

Data Type

adInteger

Modifiable

Read-only

OLE DB Property

DBPROP_NULLCOLLATION

NULL Concatenation Behavior Property

The **NULL Concatenation Behavior** property specifies how the data source handles the concatenation of NULL-valued character data type columns with non–NULL-valued character data type columns.

Applies To

Connection Object

Syntax

object.Properties("NULL Concatenation Behavior")

Parts

object

Expression that evaluates to an object in the Applies To list.

Returns

One of the following settings:

Constant	Description
DBPROPVAL_CB_NULL	The result is NULL valued.
DBPROPVAL_CB_NON_NULL	The result is the concatenation of the
	non–NULL-valued column or columns.

Data Type

adInteger

Modifiable

Read-only

OLE DB Property

DBPROP_CONCATNULLBEHAVIOR

0

Objects Transacted Property

The **Objects Transacted** property specifies whether an object created on the referenced columns can be committed in a transaction.

Applies To

Command Object Rec

Recordset Object

Syntax

object.Properties("Objects Transacted") [= value]

Parts

object

Expression that evaluates to an object in the Applies To list.

value

Boolean that specifies whether the object can be committed in a transaction, as specified in Settings.

Settings

Value	Description
True	Any object created on the referenced column can be transacted. Data made visible to the database through the object can be committed with CommitTrans or aborted with Rollback .
False	Any object created on the referenced column cannot be transacted. All changes to the object are permanent once they are made visible to the database.

adBoolean

Modifiable

Read/write

OLE DB Property

DBPROP_TRANSACTEDOBJECT

Remarks

If the **Objects Transacted** property is set on a column that does not contain an object, it is ignored.

OLE DB Version Property

The **OLE DB Version** property returns the version of OLE DB supported by the provider.

Applies To

Connection Object

Syntax

object.Properties("OLE DB Version")

Parts

object

Expression that evaluates to an object in the Applies To list.

Data Type

adBSTR

Modifiable

Read-only

OLE DB Property

DBPROP_PROVIDEROLEDBVER

Remarks

The version is of the form ##.##, where the first two digits are the major version and the next two digits are the minor version. For example, an OLE DB provider that conforms to the 2.6 specification returns "02.60".

See Also

DBMS Version Property

OLE Object Support Property

The **OLE Object Support** property returns a bitmask that specifies how the provider supports access to BLOBs and COM objects stored in columns.

Applies To

Connection Object

Syntax

object.Properties("OLE Object Support")

Parts

object

Expression that evaluates to an object in the Applies To list.

Settings

A combination of zero or more of the following settings:

Constant	Description
DBPROPVAL_OO_BLOB	The provider supports access to
	BLOBs as structured storage objects. A
	consumer determines which interfaces
	are supported using the Structured
	Storage property.
DBPROPVAL_OO_DIRECTBIND	The provider supports direct binding.
DBPROPVAL_OO_IPERSIST	The provider supports access to COM
	objects.
DBPROPVAL_OO_ROWOBJECT	The provider supports row objects.
DBPROPVAL_OO_SCOPED	Indicates that row objects implement
	IScopedOperations.

DBPROPVAL_OO_SINGLETON	The provider supports singleton
	selects. The provider supports the
	return of row objects using the
	Execute method.

adInteger

Modifiable

Read-only

OLE DB Property

DBPROP_OLEOBJECTS

Open Rowset Support Property

The **Open Rowset Support** property returns a bitmask that specifies how the provider supports opening objects through the **Connection** object.

Applies To

Connection Object

Syntax

object.Properties("Open Rowset Support")

Parts

object

Expression that evaluates to an object in the Applies To list.

Returns

A bitmask containing one or more of the following settings:

Constant	Description
DBPROPVAL_ORS_TABLE	The provider supports
	opening tables through the
	Connection object (true for
	all providers).
DBPROPVAL_ORS_INDEX	The provider supports
	specifying an index through
	the Connection object.
DBPROPVAL_ORS_INTEGRATEDINDEX	The provider supports
	specifying both a table and an
	index in the same call to the
	Execute method in order to

	open the recordset using the specified index.
DBPROPVAL_ORS_STOREDPROC	The provider supports opening a recordset over stored procedures by specifying the stored procedure name.
DBPROPVAL_ORS_HISTOGRAM	The provider supports opening a histogram recordset using the Execute method.

adInteger

Modifiable

Read-only

OLE DB Property

DBPROP_OPENROWSETSUPPORT

ORDER BY Columns in Select List Property

The **ORDER BY Columns in Select List** property indicates whether columns in an ORDER BY clause must be included in the SELECT statement.

Applies To

Connection Object

Syntax

object.Properties("ORDER BY Columns in Select List")

Parts

object

Expression that evaluates to an object in the Applies To list.

Returns

One of the following settings:

Value	Description
	Columns in an ORDER BY clause must be included in the SELECT statement.
	Columns in an ORDER BY clause are not required to be included in the SELECT statement.

Data Type

adBoolean

Modifiable

Read-only

OLE DB Property

DBPROP_ORDERBYCOLUMNSINSELECT

Others' Changes Visible Property

The **Others' Changes Visible** property specifies whether row updates or deletions by a process other than the application accessing a recordset are visible without statement reexecution.

Applies To

Command Object Recordset Object

Syntax

object.Properties("Others' Changes Visible") [= value]

Parts

object

Expression that evaluates to an object in the Applies To list.

value

Boolean that specifies whether updates or deletions by another process are visible, as described in Settings.

Settings

Value	Description	
True	Rows modified (updated or deleted) by an application or proces	
	other than the application accessing the recordset are visible. For	
	example, if another process or application updates the data	
	underlying a row or deletes the row, and the row is released	
	completely, any application accessing the recordset will see that	
	change the next time it fetches the row. This includes updates	
	and deletes made by others in the same transaction as well as	
	updates and deletes made by others outside the transaction.	

	The transaction isolation level does not affect the visibility of rows inserted by others in the same transaction, such as other recordsets in the same session. However, it does restrict the visibility of rows inserted by others outside the transaction.
False	Changes to the recordset (updates and deletes) made by other applications accessing the recordset are not visible unless the command is reexecuted.

adBoolean

Modifiable

Read/write

OLE DB Property

DBPROP_OTHERUPDATEDELETE

See Also

Others' Inserts Visible Property

Own Changes Visible Property

Own Inserts Visible Property

Others' Inserts Visible Property

The **Others' Inserts Visible** property specifies whether row inserts by a process other than the application accessing a recordset are visible without statement reexecution.

Applies To

Command Object Recordset Object

Syntax

object.Properties("Others' Inserts Visible") [= value]

Parts

object

Expression that evaluates to an object in the Applies To list.

value

Boolean that specifies whether inserts by another process are visible, as described in Settings.

Settings

Value	Description	
True	Rows inserted by an application or process other than the	
	application accessing the recordset are visible. Therefore, any	
	application accessing the recordset will see those rows the next	
	time it fetches a set of rows containing the inserted rows. This	
	includes rows inserted in the same transaction as well as rows	
	inserted outside the transaction by others.	
	The transaction isolation level does not affect the visibility of rows inserted by others in the same transaction, such as other	

recordsets in the same session. However, it does restrict the visibility of rows inserted by others outside the transaction.
Inserts to the recordset made by other applications accessing the recordset are not visible unless the command is reexecuted.

adBoolean

Modifiable

Read/write

OLE DB Property

DBPROP_OTHERINSERT

See Also

Others' Changes Visible Property Own Changes Visible Property Own Inserts Visible Property

Output Encoding Property

The **Output Encoding** property specifies the encoding to use in the stream set or returned by the **Execute** method.

Applies To

Command Object

Syntax

object.Properties("Output Encoding") [= value]

Parts

object

Expression that evaluates to an object in the Applies To list.

value

String that specifies the output encoding.

Data Type

adBSTR

Modifiable

Read/write

OLE DB Property

DBPROP_OUTPUTENCODING

Remarks

Encodings include UTF8, ANSI, and Unicode. If **Output Encoding** is not set, encoding defaults to UTF8.

If a template is specified at the URL using the **template=** keyword, the encoding is Unicode. For all other templates, the encoding is obtained from the template, which is a valid XML document and, therefore, has its own encoding.

If an XML template is specified at the URL (instead of as a Transact-SQL query) and **Output Encoding** is also specified, the encoding specified in **Output Encoding** overrides the template.

See Also

Output Stream Property

Output Parameter Availability Property

The **Output Parameter Availability** property specifies when output parameter values become available to an application.

Applies To

Connection Object

Syntax

object.Properties("Output Parameter Availability")

Parts

object

Expression that evaluates to an object in the Applies To list.

Returns

One of the following settings:

Constant	Description
DBPROPVAL_OA_NOTSUPPORTED	Output parameters are not
	supported.
DBPROPVAL_OA_ATEXECUTE	Output parameter data is available
	immediately after the Execute
	method returns.
DBPROPVAL_OA_ATROWRELEASE	If a command returns a single
	result that is a recordset, output
	parameter data is available at the
	time the recordset is completely
	released. If a command returns
	multiple results, output parameter

data is available when the
NextRecordset method returns
the next recordset or the multiple
results object is completely
released, whichever occurs first.
Before the output parameter data
is available, the consumer's bound
memory is in an indeterminate
state.

adInteger

Modifiable

Read-only

OLE DB Property

DBPROP_OUTPUTPARAMETERAVAILABILITY

Output Stream Property

The **Output Stream** property specifies the stream containing the results returned by the **Execute** method.

Applies To

Command Object

Syntax

object.Properties("Output Stream") [= value]

Parts

object

Expression that evaluates to an object in the Applies To list.

value

Pointer to the stream.

Data Type

adIUnknown

Modifiable

Read/write

OLE DB Property

DBPROP_OUTPUTSTREAM

Remarks

Use **Output Stream** to pass a reference to the stream to other processes (for example, an XML parser), thereby avoiding the overhead associated with maintaining multiple copies of the data.

See Also

Output Encoding Property

Own Changes Visible Property

The **Own Changes Visible** property specifies whether row updates or deletions by the application accessing a recordset are visible without statement reexecution.

Applies To

Command Object Record

Recordset Object

Syntax

object.Properties("Own Changes Visible") [= value]

Parts

object

Expression that evaluates to an object in the Applies To list.

value

Boolean that specifies whether updates or deletions by the application are visible, as described in Settings.

Settings

Value	Description
True	The updates and deletes made by the application accessing the
	recordset are visible. For example, if a consumer of the recordset
	updates or deletes a row, and the row is released completely, the
	update or delete will be visible to any consumer of the recordset
	the next time it fetches that row. This ability is independent of
	the transaction isolation level because all consumers of the
	recordset share the same transaction.
False	Changes to the recordset (updates and deletes) made by

applications accessing the recordset are not visible unless the command is reexecuted.

Data Type

adBoolean

Modifiable

Read/write

OLE DB Property

DBPROP_OWNUPDATEDELETE

See Also

Others' Changes Visible Property Others' Inserts Visible Property Own Inserts Visible Property

Own Inserts Visible Property

The **Own Inserts Visible** property specifies whether row inserts by the application accessing a recordset are visible without statement reexecution.

Applies To

Command Object Recordset Object

Syntax

object.Properties("Own Inserts Visible") [= value]

Parts

object

Expression that evaluates to an object in the Applies To list.

value

Boolean that specifies whether inserts by the application are visible, as described in Settings.

Settings

Value	Description
True	Inserts to the recordset are visible. If an application accessing a recordset inserts a row, that row is visible to any application
	accessing the recordset the next time the application fetches a set of rows containing that row. This ability is independent of the transaction isolation level because all applications accessing the recordset share the same transaction.
False	Changes to the recordset (updates and deletes) made by applications accessing the recordset are not visible unless the command is reexecuted.

adBoolean

Modifiable

Read/write

OLE DB Property

DBPROP_OWNINSERT

See Also

Immobile Rows Property Others' Changes Visible Property Others' Inserts Visible Property Own Changes Visible Property

P

Pass By Ref Accessors Property

The **Pass By Ref Accessors** property indicates whether the provider supports the DBACCESSOR_PASSBYREF flag in the OLE DB **IAccessor::CreateAccessor** interface. This applies both to row and to parameter accessors.

Applies To

Connection Object

Syntax

object.Properties("Pass By Ref Accessors")

Parts

object

Expression that evaluates to an object in the Applies To list.

Data Type

adBoolean

Modifiable

Read-only

OLE DB Property

DBPROP_BYREFACCESSORS

Remarks

The ADO **ActualSize** and **Value** properties map to the OLE DB **IAccessor::CreateAccessor** interface.

Password Property

The **Password** property specifies the password to be used when connecting to a data source.

Applies To

Connection Object

Syntax

object.Properties("Password") [= value]

Parts

object

Expression that evaluates to an object in the Applies To list.

value

String that contains the password.

Data Type

adBSTR

Modifiable

Read/write

OLE DB Property

DBPROP_AUTH_PASSWORD

Remarks

When the value of the **Password** property is retrieved, the provider might return a mask such as "*****" or an empty string instead of the actual password. The password is still set internally and is used when the **Open** method is called.

An ADO application can also use the OLE DB **Pwd** keyword, which is equivalent to the **Password** property.

Note The recommended method for connecting to an instance of Microsoft® SQL Server[™] 2000 is to use Windows Authentication mode.

Persist Security Info Property

The **Persist Security Info** property specifies whether the data source can persist sensitive authentication information such as a password.

Applies To

Connection Object

Syntax

object.Properties("Persist Security Info") [= value]

Parts

object

Expression that evaluates to an object in the Applies To list.

value

Boolean that specifies whether the data source can persist authentication information, as specified in Settings.

Settings

Value	Description	
True	The data source object can persist sensitive authentication	
	information such as a password along with other authentication	
	information.	
False	The data source object cannot persist sensitive authentication	
	information.	

Data Type

adBoolean

Modifiable

Read/write

OLE DB Property

DBPROP_AUTH_PERSIST_SENSITIVE_AUTHINFO

Remarks

If **Persist Security Info** is set to False at the time the data source is initialized, the data source cannot persist sensitive authentication information. Furthermore, a call to a property that contains sensitive authentication information, such as a password, returns a default value instead of the actual password.

After the data source has been uninitialized, sensitive information that was set when the data source was initialized with the **Persist Security Info** property set to False still cannot be obtained from the **Properties** collection or by persisting the uninitialized data source object. However, new properties set after the data source object has been uninitialized can be persisted or obtained if **Persist Security Info** is set to True.

Before the data source is initialized for the first time, sensitive information can be obtained from the **Properties** collection, and can be persisted, regardless of the setting of the **Persist Security Info** property. Therefore, sensitive applications should avoid passing uninitialized data source objects.

Note The recommended method for connecting to an instance of Microsoft® SQL Server[™] 2000 is to use Windows Authentication mode.

Persistent ID Type Property

The **Persistent ID Type** property specifies the type of DBID that the provider uses when persisting DBIDs that name entities in the database, such as tables, indexes, columns, commands, or constraints. This is generally the type of DBID that the provider considers the most permanent under schema changes and physical data reorganizations.

Applies To

Connection Object

Syntax

object.Properties("Persistent ID Type")

Parts

object

Expression that evaluates to an object in the Applies To list.

Returns

One of the following settings:

Constant	Description
DBPROPVAL_PT_GUID	GUID value must be provided.
DBPROPVAL_PT_GUID_NAME	GUID and _NAME values must be provided.
DBPROPVAL_PT_GUID_PROPID	GUID and _PROPID values must be provided.
DBPROPVAL_PT_NAME	NAME value must be provided.
DBPROPVAL_PT_PGUID_NAME	GUID and _NAME values must be provided.

DBPROPVAL_PT_PGUID_PROPID	GUID and _PROPID values must be
	provided.
DBPROPVAL_PT_PROPID	PROPID value must be provided.

Data Type

adInteger

Modifiable

Read-only

OLE DB Property

DBPROP_PERSISTENTIDTYPE

Prepare Abort Behavior Property

The **Prepare Abort Behavior** property indicates how aborting a transaction affects prepared commands.

Applies To

Connection Object

Syntax

object.Properties("Prepare Abort Behavior")

Parts

object

Expression that evaluates to an object in the Applies To list.

Returns

One of the following settings:

Constant	Description
DBPROPVAL_CB_DELETE	Aborting a transaction deletes prepared
	commands. The application must reprepare
	commands before executing them.
DBPROPAL_CB_PRESERVE	Aborting a transaction preserves prepared
	commands. The application can reexecute
	commands without repreparing them.

Data Type

adInteger

Modifiable

Read-only

OLE DB Property

DBPROP_PREPAREABORTBEHAVIOR

See Also

Prepare Commit Behavior Property Preserve on Abort Property Preserve on Commit Property

Prepare Commit Behavior Property

The **Prepare Commit Behavior** property specifies how committing a transaction affects prepared commands.

Applies To

Connection Object

Syntax

object.Properties("Prepare Commit Behavior")

Parts

object

Expression that evaluates to an object in the Applies To list.

value

Integer that specifies how committing a transaction affects prepared commands, as specified in Settings

Settings

One of the following settings:

Constant	Description
	Committing a transaction deletes prepared commands. The application must reprepare commands before executing them.
	Committing a transaction preserves prepared commands. The application can reexecute commands without repreparing them.

Data Type

adInteger

Modifiable

Read-only

OLE DB Property

DBPROP_PREPARECOMMITBEHAVIOR

See Also

Prepare Abort Behavior Property Preserve on Abort Property

Preserve on Commit Property

Preserve on Abort Property

The **Preserve on Abort** property specifies whether a recordset remains active after a transaction is aborted.

Applies To

		Command Object	Recordset Object
--	--	----------------	------------------

Syntax

object.Properties("Preserve on Abort") [= value]

Parts

object

Expression that evaluates to an object in the Applies To list.

value

Boolean that specifies whether a recordset remains active, as specified in Settings.

Settings

Value	Description
True	After aborting a transaction, the recordset remains active. Therefore, it is possible to fetch new rows, update, delete, and insert rows, and so on.
False	After aborting a transaction, the only operations allowed on a recordset are to release rows and the recordset.

Data Type

adBoolean

Modifiable

Read/write

OLE DB Property

DBPROP_ABORTPRESERVE

Remarks

Preserve on Abort applies only to local transactions.

See Also

Prepare Abort Behavior Property Prepare Commit Behavior Property Preserve on Commit Property

Preserve on Commit Property

The **Preserve on Commit** property specifies whether a recordset remains active after a transaction is committed.

Applies To

Syntax

object.Properties("Preserve on Commit") [= value]

Parts

object

Expression that evaluates to an object in the Applies To list.

value

Boolean that specifies whether a recordset remains active, as specified in Settings.

Settings

Value	Description
True	After committing a transaction, the recordset remains active. Therefore, it is possible to fetch new rows; update, delete, and insert rows; and so on.
False	After committing a transaction, the only operations allowed on a recordset are to release rows and the recordset.

Data Type

adBoolean

Modifiable

Read/write

OLE DB Property

DBPROP_COMMITPRESERVE

Remarks

Preserve on Commit applies only to local transactions.

See Also

Prepare Abort Behavior Property Prepare Commit Behavior Property Preserve on Abort Property Transaction DDL Property

Procedure Term Property

The **Procedure Term** property returns a character string with the database name for a procedure (for example, database procedure, stored procedure, or procedure).

Applies To

Connection Object

Syntax

object.Properties("Procedure Term")

Parts

object

Expression that evaluates to an object in the Applies To list.

Data Type

adBSTR

Modifiable

Read-only

OLE DB Property

DBPROP_PROCEDURETERM

Remarks

An application can use the value returned by the **Procedure Term** property in user interfaces.

See Also

Catalog Term Property Schema Term Property Table Term Property

Prompt Property

The **Prompt** property specifies how to prompt the user when connecting to a data source.

Applies To

Connection Object

Syntax

object.Properties("Prompt") [= value]

Parts

object

Expression that evaluates to an object in the Applies To list.

value

Small integer that indicates how to prompt the user, as specified in Settings.

Settings

Constant	Description
DBPROMPT_PROMPT	Always prompt the user for
	initialization information.
DBPROMPT_COMPLETE	Prompt the user only if more
	information is needed.
DBPROMPT_COMPLETEREQUIRED	Prompt the user only if more
	information is needed. Do not
	allow the user to enter optional
	information.
DBPROMPT_NOPROMPT	Do not prompt the user.

Data Type

adSmallInt

Modifiable

Read/write

OLE DB Property

DBPROP_INIT_PROMPT

Remarks

Information obtained from the user during prompting is available following initialization from the **Properties** collection. To prompt for initialization information, the OLE DB provider typically displays a dialog box to the user.

Dynamic properties of a **Connection** object are lost when the connection is closed. Therefore, the **Prompt** property must be reset before re-opening the connection to use a value other than the default.

Note An application should not specify that the provider should prompt the user in scenarios in which the user will not be able to respond to the dialog box. For example, the user will not be able to respond if the application is running on a server system instead of on the user's client, or if the application is running on a system with no user logged on. In these cases, the application will wait indefinitely for a response and appear to lock up.

Provider Friendly Name Property

The **Provider Friendly Name** property returns the display name of the provider (for example, "Microsoft OLE DB Provider for SQL Server").

Applies To

Connection Object

Syntax

object.Properties("Provider Friendly Name")

Parts

object

Expression that evaluates to an object in the Applies To list.

Data Type

adBSTR

Modifiable

Read-only

OLE DB Property

DBPROP_PROVIDERFRIENDLYNAME

See Also

Provider Name Property

Provider Version Property

Provider Name Property

The **Provider Name** property returns the file name of the provider (for example, Sqloledb.dll).

Applies To

Connection Object

Syntax

object.Properties("Provider Name")

Parts

object

Expression that evaluates to an object in the Applies To list.

Data Type

adBSTR

Modifiable

Read-only

OLE DB Property

DBPROP_PROVIDERFILENAME

See Also

Provider Friendly Name Property

Provider Version Property

Provider Version Property

The **Provider Version** property returns the version of the provider.

Applies To

Connection Object

Syntax

object.Properties("Provider Version")

Parts

object

Expression that evaluates to an object in the Applies To list.

Data Type

adBSTR

Modifiable

Read-only

OLE DB Property

DBPROP_PROVIDERVER

Remarks

The version is of the form ##.##.####, where the first two digits are the major version, the next two digits are the minor version, and the last four digits are the release version. The provider can append a description of the provider.

The **Provider Version** property is equivalent to the **DBMS Version** property if

the DBMS supports OLE DB interfaces directly. It is different if the provider is separate from the DBMS, such as when the provider accesses the DBMS through ODBC.

See Also

Provider Friendly Name Property

Provider Name Property

Q

Quick Restart Property

The **Quick Restart** property specifies whether the command that created a recordset must be reexecuted before the **MoveFirst** method is executed.

Applies To

Command Object Recordset Object

Syntax

object.Properties("Quick Restart") [= value]

Parts

object

Expression that evaluates to an object in the Applies To list.

value

Boolean that indicates whether the command that created a recordset must be reexecuted, as specified in Settings.

Settings

Value	Description
	The MoveFirst method is relatively quick to execute. In particular, it does not reexecute the command that created the recordset.
	The MoveFirst method is expensive to execute and requires reexecuting the command that created the recordset.

Data Type

adBoolean

Modifiable

Read/write

OLE DB Property

DBPROP_QUICKRESTART

Remarks

Although the value of this property can be set to True, the provider is not required to honor it. The reason for this is that the provider does not know what the command is at the time the property is set. For example, the application can set this property and then change the command text. However, the provider can fail the **Quick Restart** property if it is never able to quickly restart the next fetch position. Therefore, if an application successfully sets the **Quick Restart** property, it must still check this flag on the recordset to determine if the next fetch position can be quickly set.

R

Read-Only Data Source Property

The **Read-Only Data Source** property indicates whether the referenced database is read-only.

Applies To

Connection Object

Syntax

object.Properties("Read-Only Data Source")

Parts

object

Expression that evaluates to an object in the Applies To list.

Returns

One of the following settings:

Value	Description
True	The database is read-only.
False	The database is updatable.

Data Type

adBoolean

Modifiable

Read-only

OLE DB Property

DBPROP_DATASOURCEREADONLY

Reentrant Events Property

The **Reentrant Events** property indicates whether the provider supports reentrancy on **Recordset** methods.

Applies To

Command Object Recordset Object

Syntax

object.Properties("Reentrant Events")

Parts

object

Expression that evaluates to an object in the Applies To list.

Returns

One of the following Settings:

Value	Description
True	The provider supports reentrancy during callbacks to the OLE
	DB IRowsetNotify interface. The provider might not support
	reentrancy on all Recordset methods. These methods return
	DB_E_NOTREENTRANT.
False	The provider does not support such reentrancy. The provider
	returns DB_E_NOTREENTRANT on methods called during the
	notification.

Data Type

adBoolean

Modifiable

Read-only

OLE DB Property

DBPROP_REENTRANTEVENTS

Remarks

Regardless of how the **Reentrant Events** property is set, all providers support **GetRows** and **Close** methods during notifications, as long as the columns being accessed do not include deferred columns.

Remove Deleted Rows Property

The **Remove Deleted Rows** property specifies whether rows that are detected as deleted are removed from the recordset.

Applies To

Command Object

Recordset Object

Syntax

object.Properties("Remove Deleted Rows") [= value]

Parts

object

Expression that evaluates to an object in the Applies To list.

value

Boolean that indicates whether rows are removed from the recordset, as specified in Settings.

Settings

Value	Description
True	The provider removes rows it detects as having been deleted
	from the recordset. Therefore, fetching a block of rows that
	formerly included a deleted row does not return that row.
False	The provider deletes the rows, but does not remove them from
	the recordset. If the user fetches a block of rows containing a
	deleted row, that row appears in the recordset.
	Any method that retrieves a deleted row will return a code of DB_E_DELETEDROW.

Data Type

adBoolean

Modifiable

Read/write

OLE DB Property

DBPROP_REMOVEDELETED

Remarks

This property is independent of the transaction isolation level. While the transaction isolation level in some cases determines whether the recordset can detect a row as having been deleted, it has no effect on whether or not the recordset removes that row.

For programmers accustomed to the cursor model in ODBC, the value of this property is always True for recordsets implemented using dynamic cursors because dynamic cursors always remove deleted rows. Whether static and keyset-driven cursors remove deleted rows depends on the value of the **Remove Deleted Rows** property.

Report Multiple Changes Property

The **Report Multiple Changes** property indicates whether an update or delete operation can affect multiple rows.

Applies To

Command Object

Recordset Object

Syntax

object.Properties("Report Multiple Changes")

Parts

object

Expression that evaluates to an object in the Applies To list.

Returns

One of the following settings:

Value	Description
True	An update or delete operation can affect multiple rows, and the
	provider can detect that multiple rows have been updated or
	deleted. This happens when a provider cannot uniquely identify a
	row. For example, the provider might use the values of all the
	columns in the row to identify the row; if these columns do not
	include a unique key, an update or delete might affect more than
	one row.
False	An update or delete always affects a single row, or the provider
	cannot detect whether it affects multiple rows.

Data Type

adBoolean

Modifiable

Read-only

OLE DB Property

DBPROP_REPORTMULTIPLECHANGES

Return Pending Inserts Property

The **Return Pending Inserts** property indicates whether pending insert rows can be returned.

Applies To

Command Object	Recordset Object	
----------------	------------------	--

Syntax

object.Properties("Return Pending Inserts")

Parts

object

Expression that evaluates to an object in the Applies To list.

Returns

One of the following settings:

Value	Description
True	The methods that fetch rows, such as GetRows , Move ,
	MoveNext, and MovePrevious can return pending insert rows
	(rows that have been inserted in delayed update mode but for
	which the Update method has not yet been called).
False	The methods that fetch rows cannot return pending insert rows.

Data Type

adBoolean

Modifiable

Read-only

OLE DB Property

DBPROP_RETURNPENDINGINSERTS

Row Delete Notification Property

The **Row Delete Notification** property returns a bitmask that indicates whether the notification phase is cancelable.

Applies To

Command Object Recordset Object

Syntax

object.Properties("Row Delete Notification")

Parts

object

Expression that evaluates to an object in the Applies To list.

Returns

Zero or more of the following settings:

- DBPROPVAL_NP_ABOUTTODO
- DBPROPVAL_NP_OKTODO
- DBPROPVAL_NP_SYNCHAFTER

Data Type

adInteger

Modifiable

Read-only

OLE DB Property

DBPROP_NOTIFYROWDELETE

Row First Change Notification Property

The **Row First Change Notification** property returns a bitmask specifying whether the notification phase is cancelable.

Applies To

Command Object

Recordset Object

Syntax

object.Properties("Row First Change Notification")

Parts

object

Expression that evaluates to an object in the Applies To list.

Returns

Zero or more of the following settings:

- DBPROPVAL_NP_ABOUTTODO
- DBPROPVAL_NP_OKTODO
- DBPROPVAL_NP_SYNCHAFTER

Data Type

adInteger

OLE DB Property

DBPROP_NOTIFYROWFIRSTCHANGE

Row Insert Notification Property

The **Row Insert Notification** property returns a bitmask specifying whether the notification phase is cancelable.

Applies To

Command Object Recordset Object

Syntax

object.Properties("Row Insert Notification")

Parts

object

Expression that evaluates to an object in the Applies To list.

Returns

Zero or more of the following settings:

- DBPROPVAL_NP_ABOUTTODO
- DBPROPVAL_NP_OKTODO
- DBPROPVAL_NP_SYNCHAFTER

Data Type

adInteger

OLE DB Property

DBPROP_NOTIFYROWINSERT

Row Privileges Property

The **Row Privileges** property indicates whether access rights are restricted on a row-by-row basis.

Applies To

Command Object	Recordset Object
<u></u>	

Syntax

object.Properties("Row Privileges")

Parts

object

Expression that evaluates to an object in the Applies To list.

Returns

One of the following values:

Value	Description
True	Access rights are restricted on a row-by-row basis. If the recordset
	supports the OLE DB IRowsetChange interface, the ADO
	Update method can be called for some but not all rows. A
	recordset must never count or return a handle for a row for which
	the application does not have read access rights.
False	Access rights are not restricted on a row-by-row basis. If the
	recordset supports the OLE DB IRowsetChange interface, the
	ADO Update method can be called for any row.

Data Type

adBoolean

Modifiable

Read-only

OLE DB Property

DBPROP_ROWRESTRICT

See Also

Column Privileges Property

Row Resynchronization Notification Property

The **Row Resynchronization Notification** property returns a bitmask that specifies whether the notification phase is cancelable.

Applies To

Command Object

Recordset Object

Syntax

object.Properties("Row Resynchronization Notification")

Parts

object

Expression that evaluates to an object in the Applies To list.

Returns

Zero or more of the following settings:

- DBPROPVAL_NP_ABOUTTODO
- DBPROPVAL_NP_OKTODO
- DBPROPVAL_NP_SYNCHAFTER

Data Type

adInteger

OLE DB Property

DBPROP_NOTIFYROWRESYNCH

Row Threading Model Property

The **Row Threading Model** property specifies which threading models are supported by the rowset.

Applies To

Command Object Recordset Object

Syntax

object.Properties("Row Threading Model") [= value]

Parts

object

Expression that evaluates to an object in the Applies To list.

value

Integer that indicates which threading models are supported by the rowset, as specified in Settings.

Settings

A bitmask containing one or more of the following settings:

Constant	Description
DBPROPVAL_RT_APTMTTHREAD	The apartment thread model is
	supported.
DBPROPVAL_RT_FREETHREAD	The free thread model is supported.
DBPROPVAL_RT_SINGLETHREAD	The single thread model is
	supported.

Data Type

adInteger

Modifiable

Read/write

OLE DB Property

DBPROP_ROWTHREADMODEL

Row Undo Change Notification Property

The **Row Undo Change Notification** property returns a bitmask specifying whether the notification phase is cancelable.

Applies To

Command Object

Recordset Object

Syntax

object.Properties("Row Undo Change Notification")

Parts

object

Expression that evaluates to an object in the Applies To list.

Returns

Zero or more of the following settings:

- DBPROPVAL_NP_ABOUTTODO
- DBPROPVAL_NP_OKTODO
- DBPROPVAL_NP_SYNCHAFTER

Data Type

adInteger

OLE DB Property

DBPROP_NOTIFYROWUNDOCHANGE

Row Undo Delete Notification Property

The **Row Undo Delete Notification** property returns a bitmask specifying whether the notification phase is cancelable.

Applies To

Command Object

Recordset Object

Syntax

object.Properties("Row Undo Delete Notification")

Parts

object

Expression that evaluates to an object in the Applies To list.

Returns

Zero or more of the following settings:

- DBPROPVAL_NP_ABOUTTODO
- DBPROPVAL_NP_OKTODO
- DBPROPVAL_NP_SYNCHAFTER

Data Type

adInteger

OLE DB Property

DBPROP_NOTIFYROWUNDODELETE

Row Undo Insert Notification Property

The **Row Undo Insert Notification** property returns a bitmask specifying whether the notification phase is cancelable.

Applies To

Command Object

Recordset Object

Syntax

object.Properties("Row Undo Insert Notification")

Parts

object

Expression that evaluates to an object in the Applies To list.

Returns

Zero or more of the following settings:

- DBPROPVAL_NP_ABOUTTODO
- DBPROPVAL_NP_OKTODO
- DBPROPVAL_NP_SYNCHAFTER

Data Type

adInteger

OLE DB Property

DBPROP_NOTIFYROWUNDOINSERT

Row Update Notification Property

The **Row Update Notification** property returns a bitmask specifying whether the notification phase is cancelable.

Applies To

Command Object Recordset Object

Syntax

object.Properties("Row Update Notification")

Parts

object

Expression that evaluates to an object in the Applies To list.

Returns

Zero or more of the following settings:

- DBPROPVAL_NP_ABOUTTODO
- DBPROPVAL_NP_OKTODO
- DBPROPVAL_NP_SYNCHAFTER

Data Type

adInteger

OLE DB Property

DBPROP_NOTIFYROWUPDATE

Rowset Conversions on Command Property

The **Row Conversions on Command** property specifies how inquiries on a command about supported conversions are handled.

Applies To

Connection Object

Syntax

object.Properties("Rowset Conversions on Command")

Parts

object

Expression that evaluates to an object in the Applies To list.

Settings

Value	Description	
True	Callers to the OLE DB IConvertType::CanConvert interface	
	can inquire on a command about conversions supported on	
	recordsets generated by the command.	
False	Callers can inquire on a command only about conversions	
	supported by the command.	

Data Type

adBoolean

OLE DB Property

DBPROP_ROWSETCONVERSIONSONCOMMAND

Rowset Fetch Position Change Notification Property

The **Rowset Fetch Position Change Notification** property returns a bitmask specifying whether the notification phase of DBREASON_ROWSET_CHANGE is cancelable.

Applies To

Command Object

Recordset Object

Syntax

object.Properties("Rowset Fetch Position Change Notification")

Parts

object

Expression that evaluates to an object in the Applies To list.

Returns

Zero or more of the following settings:

- DBPROPVAL_NP_ABOUTTODO
- DBPROPVAL_NP_OKTODO
- DBPROPVAL_NP_SYNCHAFTER

Data Type

adInteger

OLE DB Property

DBPROP_NOTIFYROWSETFETCHPOSITIONCHANGE

Rowset Release Notification Property

The **Rowset Release Notification** property returns a bitmask specifying whether the notification phase is cancelable.

Applies To

Command Object

Recordset Object

Syntax

object.Properties("Rowset Release Notification")

Parts

object

Expression that evaluates to an object in the Applies To list.

Returns

Zero or more of the following settings:

- DBPROPVAL_NP_ABOUTTODO
- DBPROPVAL_NP_OKTODO
- DBPROPVAL_NP_SYNCHAFTER

Data Type

adInteger

OLE DB Property

DBPROP_NOTIFYROWSETRELEASE

S

Schema Term Property

The **Schema Term** property returns the name the data source uses for a schema (for example, schema or owner).

Applies To

Connection Object

Syntax

object.Properties("Schema Term")

Parts

object

Expression that evaluates to an object in the Applies To list.

Data Type

adBSTR

Modifiable

Read-only

OLE DB Property

DBPROP_SCHEMATERM

Remarks

An application can use the value returned by the **Schema Term** property in user interfaces.

See Also

Catalog Term Property Procedure Term Property Table Term Property

Schema Usage Property

The **Schema Usage** property returns a bitmask specifying how schema names can be used in text commands.

Applies To

Connection Object

Syntax

object.Properties("Schema Usage")

Parts

object

Expression that evaluates to an object in the Applies To list.

Returns

A combination of zero or more of the following settings:

Constant	Description
DBPROPVAL_SU_DML_STATEMENTS	Schema names are supported in all data manipulation language (DML) statements.
DBPROPVAL_SU_TABLE_DEFINITION	Schema names are supported in all table definition statements.
DBPROPVAL_SU_INDEX_DEFINITION	Schema names are supported in all index definition statements and may apply only to the

	table name, not the index
	name, depending on the
	SQL implementation.
DBPROPVAL_SU_PRIVILEGE_DEFINITION	Schema names are
	supported in all privilege
	definition statements.

Data Type

adInteger

Modifiable

Read-only

OLE DB Property

DBPROP_SCHEMAUSAGE

Scroll Backwards Property

The **Scroll Backwards** property indicates whether the recordset can scroll backward.

Applies To

Command Object Recordset Object

Syntax

object.Properties("Scroll Backwards") [= value]

Parts

object

Expression that evaluates to an object in the Applies To list.

value

Boolean that indicates whether the recordset can scroll backward, as specified in Settings.

Settings

Value	Description	
True	The recordset can scroll backward.	
False	The recordset cannot scroll backward.	

Data Type

adBoolean

Modifiable

Read/write

OLE DB Property

DBPROP_CANSCROLLBACKWARDS

Server Cursor Property

The **Server Cursor** property determines where a cursor, if required, is materialized.

Applies To

Command Object Recordset Object

Syntax

object.Properties("Server Cursor") [= value]

Parts

object

Expression that evaluates to an object in the Applies To list.

value

Boolean that determines whether a cursor is materialized on the server or on the client, as specified in Settings.

Settings

Value	Description
True	The provider attempts to support any requested cursor functionality by materializing a cursor on the server.
False	The provider attempts to support any requested cursor functionality by materializing a cursor on the client.

Data Type

adBoolean

Modifiable

Read/write

OLE DB Property

DBPROP_SERVERCURSOR

Server Data on Insert Property

The **Server Data on Insert** property specifies whether an application can retrieve values from the database for newly inserted rows.

Applies To

Command Object

Syntax

object.Properties("Server Data on Insert") [= value]

Parts

object

Expression that evaluates to an object in the Applies To list.

value

Boolean that indicates whether an application can retrieve new added values, as specified in Settings.

Settings

Value	Description
True	After an insert is transmitted to the server (when the AddNew method is called in immediate mode or when the Update method is called for an inserted row in deferred update mode), the application can call the GetRows method to retrieve the actual values that appeared in the database, including calculated columns and defaults not explicitly set in the call to AddNew .
False	The provider does not retrieve values from the database for newly inserted rows. The application can retrieve only data values explicitly set in the call to the AddNew method or by calls

Data Type

adBoolean

Modifiable

Read/write

OLE DB Property

DBPROP_SERVERDATAONINSERT

Remarks

Setting the **Server Data on Insert** property is potentially expensive and may not be supported for certain types of recordsets.

Skip Deleted Bookmarks Property

The **Skip Deleted Bookmarks** property indicates whether the recordset allows certain methods of the **RecordSet** object to skip a bookmark row and continue with the next row.

Applies To

<u>Command Object</u>

Recordset Object

Syntax

object.Properties("Skip Deleted Bookmarks") [= value]

Parts

object

Expression that evaluates to an object in the Applies To list.

value

Boolean that indicates whether the called method skips the bookmark row, as specified in Settings.

Settings

Value	Description	
True	The called method skips the bookmark row and continues with	
	the next row.	
False	The called method returns DB_E_BADBOOKMARK.	

Data Type

adBoolean

Modifiable

Read/write

OLE DB Property

DBPROP_BOOKMARKSKIPPED

Remarks

The **Skip Deleted Bookmarks** property has impact if a bookmark row has been deleted, if a recordset contains a bookmarked row to which the user does not have access rights, a bookmark identifying a row not in the chapter, or contains a bookmarked row that is no longer a member of the recordset.

These methods are affected:

- GetRows Method
- Move Method
- MoveFirst Method
- MoveLast Method
- MoveNext Method
- **MovePrevious** Method

SQL Support Property

The **SQL Support** property returns a bitmask specifying the level of support for SQL.

Applies To

Connection Object

Syntax

object.Properties("SQL Support")

Parts

object

Expression that evaluates to an object in the Applies To list.

Returns

Zero or more of the following settings:

Constant	Description
DBPROPVAL_SQL_NONE	SQL is not supported.
DBPROPVAL_SQL_ODBC_MINIMUM	The provider supports the mini capabilities ODBC by setting t DBPROPVAL_SQL_ODBC_M bit.
DBPROPVAL_SQL_ODBC_CORE	The provider supports the core capabilities ODBC by setting to DBPROPVAL_SQL_ODBC_C
DBPROPVAL_SQL_ODBC_EXTENDED	These levels correspond to the SQL conformance defined in C version 2.5. These levels are cu That is, if the provider supports

	it also sets the bits for all lower For example, if the provider se DBPROPVAL_SQL_ODBC_C it also sets the DBPROPVAL_SQL_ODBC_N bit.
DBPROPVAL_SQL_ESCAPECLAUSES	The provider supports the ODE clause syntax.
DBPROPVAL_SQL_ANSI92_ENTRY	The provider supports the entry SQL 92.
DBPROPVAL_SQL_FIPS_TRANSITIONAL	The provider supports the trans level of the FIPS 127-2 standar
DBPROPVAL_SQL_ANSI92_INTERMEDIATE	The provider supports the inter level of SQL 92.
DBPROPVAL_SQL_ANSI92_FULL	These levels correspond to the ANSI SQL-92. These levels an cumulative. That is, if the prov supports one level, it also sets t all lower levels.
DBPROPVAL_SQL_ANSI89_IEF	The provider supports the ANS Integrity Enhancement Facility
DBPROPVAL_SQL_SUBMINIMUM	The provider supports the DBC dialect and parses the comman according to SQL rules but doe support either the minimum OI or the ANSI SQL-92 Entry lev level is not cumulative; provide support at least the minimal OI or ANSI SQL-92 Entry Level of this bit. OLE DB consumers ca determine whether or not the p supports the DBGUID_SQL di verifying that the DBPROPVAL_SQL_NONE bi set.

Data Type

adInteger

Modifiable

Read-only

OLE DB Property

DBPROP_SQLSUPPORT

SS STREAM FLAGS Property

The **SS STREAM FLAGS** property specifies how an application controls mapping schemas, XSL files, and templates.

Applies To

Command Object

Syntax

object.Properties("SS STREAM FLAGS") [= value]

Parts

object

Expression that evaluates to an object in the Applies To list.

value

Integer that specifies how an application manages mapping schemas, XSL files, and templates, as specified in Settings.

Settings

An application can use an **OR** logical operator to specify more than a single value.

Constant	Value	Description
STREAM_FLAGS_DISALLOW_URL	1	URLs are
		not accepted
		for mapping
		schemas or
		XSL. This is
		a security
		provision

		that prevents template schema references from allowing URLs to be XSL values.
STREAM_FLAGS_DISALLOW_ABSOLUTE_PATH	2	Paths specified for mapping schemas or XSL must be relative to the base path of the template itself.
STREAM_FLAGS_DISALLOW_QUERY	4	Queries are not allowed in a template. This limits the data that can be returned from the server.
STREAM_FLAGS_DONTCACHEMAPPINGSCHEMA	8	Mapping schema is not cached. This is useful during the development phase, when database

		schemas are subject to alteration.
STREAM_FLAGS_DONTCACHETEMPLATE	16	Templates are not cached. This is useful during the development phase, when database schemas are subject to alteration.
STREAM_FLAGS_DONECACHEXSL	32	XSL is not cached. This is useful during the development phase, when database schemas are subject to alteration.

Data Type

adInteger

Modifiable

Read/write

OLE DB Property

SSPROP_STREAM_FLAGS

See Also

Base Path Property Content Type Property Mapping Schema Property XML Root Property XSL Property

Strong Row Identity Property

The **Strong Row Identity** property indicates whether the handles of newly inserted rows can be compared.

Applies To

Command Object Recordset Object

Syntax

object.Properties("Strong Row Identity")

Parts

object

Expression that evaluates to an object in the Applies To list.

Returns

One of the following settings:

Value	Description
True	The handles of newly inserted rows can be compared as specified
	by the Literal Row Identity property.
False	There is no guarantee that the handles of newly inserted rows can
	be compared successfully.

Data Type

adBoolean

Modifiable

Read-only

OLE DB Property

DBPROP_STRONGIDENTITY

Remarks

A newly inserted row is defined as a row for which an insertion has been transmitted to the data source object, as opposed to a pending insert row.

See Also

Literal Row Identity Property

Structured Storage Property

The **Structured Storage** property returns a bitmask specifying which OLE DB interfaces the recordset supports on storage objects.

Applies To

Connection Object

Syntax

object.Properties("Structured Storage")

Parts

object

Expression that evaluates to an object in the Applies To list.

Returns

A combination of zero or more of the following settings:

Constant	Description
DBPROPVAL_SS_ISEQUENTIALSTREAM	The provider supports the
	OLE DB
	ISequentialStream
	interface.
DBPROPVAL_SS_ISTREAM	The provider supports the
	OLE DB IStream interface.
DBPROPVAL_SS_ISTORAGE	The provider supports the
	OLE DB IStorage interface.
DBPROPVAL_SS_ILOCKBYTES	The provider supports the
	OLE DB ILockBytes
	interface.

Data Type

adInteger

Modifiable

Read-only

OLE DB Property

DBPROP_STRUCTUREDSTORAGE

Remarks

If a provider can support any of these OLE DB interfaces, it is also required to support **ISequentialStream**.

Subquery Support Property

The **Subquery Support** property returns a bitmask specifying the predicates in text commands that support subqueries.

Applies To

Connection Object

Syntax

object.Properties("Subquery Support")

Parts

object

Expression that evaluates to an object in the Applies To list.

Returns

A combination of zero or more of the following settings:

Constant	Description
DBPROPVAL_SQ_CORRELATEDSUBQUERIES	All predicates that
	support subqueries
	support correlated
	subqueries.
DBPROPVAL_SQ_COMPARISON	Comparison operators
	are supported.
DBPROPVAL_SQ_EXISTS	The EXISTS clause is
	supported.
DBPROPVAL_SQ_IN	The IN clause is
	supported.
DBPROPVAL_SQ_QUANTIFIED	Quantified predicates

	are supported.
DBPROPVAL_SQ_TABLE	Subqueries are
	supported in place of
	tables (for example, in
	the FROM clause of an
	SQL statement).

Data Type

adInteger

Modifiable

Read-only

OLE DB Property

DBPROP_SUBQUERIES

Т

Table Term Property

The **Table Term** property returns the name the data source uses for a table (for example, table or file).

Applies To

Connection Object

Syntax

object.Properties("Table Term")

Parts

object

Expression that evaluates to an object in the Applies To list.

Data Type

adBSTR

Modifiable

Read-only

OLE DB Property

DBPROP_TABLETERM

Remarks

An application can use the value returned by the **Table Term** property in user interfaces.

See Also

Catalog Term Property
Procedure Term Property
Schema Term Property

Transaction DDL Property

The **Transaction DDL** property indicates the relationship of transactions to table and index modification data definition language (DDL) statements.

Applies To

Connection Object

Syntax

object.Properties("Transaction DDL")

Parts

object

Expression that evaluates to an object in the Applies To list.

Returns

One of the following settings:

Constant	Description
DBPROPVAL_TC_NONE	Transactions are not supported.
DBPROPVAL_TC_DML	Transactions can contain only data manipulation language (DML) statements. Attempting to modify tables or indexes within a transaction causes an error.
DBPROPVAL_TC_DDL_COMMIT	Transactions can contain only DML statements. Modifying tables or indexes within a transaction causes the transaction to be committed. The provider's commit mode remains

	unchanged in accordance with the value of the Preserve on Commit property. If the provider is in auto- commit mode, it remains in auto- commit mode. If the provider is in manual-commit mode, it remains in manual-commit mode.
DBPROPVAL_TC_DDL_IGNORE	Transactions can contain only DML statements. Attempts to modify tables or indexes within a transaction are ignored. DDL operations are not transacted even if a session is participating in a transaction. If the DDL method succeeds, the operation is complete and unaffected by subsequent calls to abort or commit the transaction.
DBPROPVAL_TC_DDL_LOCK	Transactions can contain both DML and table or index modifications, but modifying a table or index within a transaction causes the table or index to be locked until the transaction completes.
DBPROPVAL_TC_ALL	Transactions can contain DML statements, as well as table or index modifications, in any order.

Data Type

adInteger

Modifiable

Read-only

OLE DB Property

DBPROP_SUPPORTEDTXNDDL

See Also

Preserve on Commit Property

U

Unique Rows Property

The **Unique Rows** property specifies whether each row is uniquely identified by its column values.

Applies To

Recordset Object

Syntax

object.Properties("Unique Rows") [= value]

Parts

object

Expression that evaluates to an object in the Applies To list.

value

Boolean that indicates whether each row is uniquely identified, as specified in Settings.

Settings

Value	Description
True	Each row is uniquely identified by its column values.
	Rows in the recordset may or may not be uniquely identified by their column values.

Data Type

adBoolean

Modifiable

Read/write

OLE DB Property

DBPROP_UNIQUEROWS

Remarks

If the **Unique Rows** property is set to True when opening the recordset, the provider adds additional columns, if necessary, to ensure that each row is uniquely identified by its values. These additional columns appear at the end of the recordset; have a DBID of type DBKIND_GUID_PROPID, DBKIND_PGUID_PROPID, DBKIND_GUID_NAME, or DBKIND_PGUID_NAME; and the *guid* (or *pguid*) element is (or points to) DBCOL_SPECIALCOL. These columns typically are not displayed to the user but are used by components such as update services to uniquely identify a row.

The provider optionally may duplicate existing columns in the recordset to ensure that key columns are included.

If the **Unique Rows** property is set to True and the provider supports the optional OLE DB **IColumnsRowset** meta data column DBCOLUMN_KEYCOLUMN, the set of columns that uniquely identify the row have a value of True in the DBCOLUMN_KEYCOLUMN column returned by **IColumnsRowset**. This may be a subset of the columns in the row, or all of the columns if the provider cannot determine a proper subset that uniquely identifies the row.

Updatability Property

The **Updatability** property specifies the supported methods on the **Recordset** object.

Applies To

Command Object Recordset Object

Syntax

```
object.Properties("Updatability") [= value]
```

Parts

object

Expression that evaluates to an object in the Applies To list.

value

Integer that specifies supported methods, as specified in Settings.

Settings

Constant	Description
DBPROPVAL_UP_CHANGE	The Update method is supported.
DBPROPVAL_UP_DELETE	The Delete method is supported.
DBPROPVAL_UP_INSERT	The AddNew method is supported.

Data Type

adInteger

Modifiable

Read/write

OLE DB Property

DBPROP_UPDATABILITY

Remarks

The **Updatability** property should be used in conjunction with the **IRowsetChange** property. If the **IRowsetChange** property is set to True and **Updatability** is not set, the provider determines which methods are supported by the **IRowsetChange** property.

The **Updatability** property is considered successfully set if all of the bits specified are supported. It is not necessary for the provider to disable support for the bits that are not set, although providers may do so to optimize performance.

See Also

IRowsetChange Property IRowsetUpdate Property

Use Bookmarks Property

The **Use Bookmarks** property indicates whether the recordset supports bookmarks.

Applies To

Command Object	Recordset Object
----------------	------------------

Syntax

object.Properties("Use Bookmarks") [= value]

Parts

object

Expression that evaluates to an object in the Applies To list.

value

Boolean that indicates bookmark support, as specified in Settings.

Settings

Value	Description
True	The recordset supports bookmarks. Column zero is the bookmark for the rows. This column obtains a bookmark value, which can be used to reposition to the row.
False	The recordset does not support bookmarks. The recordset is sequential, and the values of the Literal Bookmarks property is ignored.

Data Type

adBoolean

Modifiable

Read/write

OLE DB Property

DBPROP_BOOKMARKS

Remarks

The value of this property is automatically set to True if the value of the **Bookmarkable** or **Literal Bookmarks** property is set to True.

See Also

Bookmarkable Property

Literal Bookmarks Property

User ID Property

The **User ID** property specifies the user ID to use when connecting to the data source.

Applies To

Connection Object

Syntax

object.Properties("User ID") [= value]

Parts

object

Expression that evaluates to an object in the Applies To list.

value

String containing the user ID.

Data Type

adBSTR

Modifiable

Read/write

OLE DB Property

DBPROP_AUTH_USERID

Remarks

An ADO application can also use the OLE DB **UID** keyword, which is equivalent to the **User ID** property.

Note The recommended method for connecting to an instance of Microsoft® SQL Server[™] 2000 is Windows Authentication mode.

User Name Property

The **User Name** property returns the name used in a particular database, which can be different than a login name.

Applies To

Connection Object

Syntax

object.Properties("User Name")

Parts

object

Expression that evaluates to an object in the Applies To list.

Data Type

adBSTR

Modifiable

Read-only

OLE DB Property

DBPROP_USERNAME

W

Window Handle Property

The **Window Handle** property specifies the window handle to use if the data source needs to prompt for additional information.

Applies To

Connection Object

Syntax

object.Properties("Window Handle") [= value]

Parts

object

Expression that evaluates to an object in the Applies To list.

value

Integer that identifies the window handle.

Data Type

adInteger

Modifiable

Read/write

OLE DB Property

DBPROP_INIT_HWND

X

XML Root Property

The **XML Root** property provides a root tag in which the query result is wrapped to return a well-formed document.

Applies To

Command Object

Syntax

object.Properties("XML Root") [= value]

Parts

object

Expression that evaluates to an object in the Applies To list.

value

String that contains the root tag.

Data Type

adBSTR

Modifiable

Read/write

OLE DB Property

SSPROP_STREAM_XMLROOT

Remarks

FOR XML and XPath queries return results in the form of document fragments, which cannot be loaded into a browser. Use the **XML Root** property to wrap the result set so that it can be loaded into DOM, or viewed in a browser. The result returns the XML Declaration, <?xml version="1.0"?>, in the output.

SQL ISAPI supports the keyword 'root', which maps to the **XML Root** property.

See Also

Base Path Property Content Type Property Mapping Schema Property SS STREAM FLAGS Property URL Access XSL Property

XSL Property

The **XSL** property specifies an XSL file name or URL applied to the result of a query.

Applies To

Command Object

Syntax

object.Properties("XSL") [= value]

Parts

object

Expression that evaluates to an object in the Applies To list.

value

String that specifies the file name or URL.

Data Type

adBSTR

Modifiable

Read/write

OLE DB Property

SSPROP_STREAM_XSL

Remarks

Relative paths are resolved by the **Base Path** property. If the **Base Path** property is not set, the relative path defaults to the current directory.

Output from command execution is expected to be a valid XML document (for example, SELECT FOR XML queries, templates, and XPath queries).

By definition, XSL takes two XML documents and produces a third. One of the input documents contains the data, and the other contains the XSL processing instructions.

Unless specified in the XSL document, the output document has a default encoding of UTF-8. If another encoding is required, it should be specified in the XSL document.

If the **Output Encoding** property is specified and an XSL document is also specified by the **XSL** property, the encoding specified in **Output Encoding** overrides the encoding of the XSL document.

See Also

Base Path Property Content Type Property Mapping Schema Property SS STREAM FLAGS Property URL Access

XML Root Property

Provider Support for ADOX

Microsoft ActiveX Data Objects Extensions for Data Definition Language and Security (ADOX) is an extension to the ADO objects and programming model. ADOX includes objects for schema creation and modification, as well as security.

The Microsoft OLE DB Provider for SQL Server (SQLOLEDB) and the Microsoft OLE DB Provider for ODBC (MSDASQL) support most ADOX features. However support for some features is restricted or unavailable.

Object or Collection	Usage Restriction
Catalog object	The Create method is not supported.
Table object	Properties are read/write prior to object creation, and read-only when referencing an existing object.
Views collection	Views is not supported.
Procedures collection	The Append and Delete methods are not supported.
Procedure object	The Command property is not supported.
Keys collection	The Append and Delete methods are not supported.
Users collection	Users is not supported.
Groups collection	Groups is not supported.

Restrictions on SQLOLEDB support for ADOX

Restrictions on MSDASQL support for ADOX

Object or Collection	Usage Restriction
Catalog object	The Create method is not supported.
Table object	Properties are read/write prior to object creation, and read-only when referencing an existing object.

Tables collection	The Append and Delete methods are not supported.
Procedures collection	The Append and Delete methods are not supported.
Procedure object	The Command property is not supported.
Indexes collection	The Append and Delete methods are not supported.
Keys collection	The Append and Delete methods are not supported.
Users collection	Users is not supported.
Groups collection	Groups is not supported.

ADO Samples

Microsoft® SQL Server[™] 2000 includes the following query applications to introduce you to using ADO:

Sample	Description
ADO and Long Data	C++ language sample. Demonstrates how to use
<u>Types (C++)</u>	ADO to display long data types.
ADO and Long Data	Visual Basic sample. Demonstrates how to use
<u>Types (Visual Basic)</u>	ADO to display long data types.
ADO Connection and	Visual Basic sample. Uses ADO to connect to an
Error Handling	instance of SQL Server.
ADO and Long Data	Web sample. Demonstrates how to use ADO to
<u>Types (Web)</u>	display long data types.
ADO Web Application	Web sample. Demonstrates how to create a Web
	application using ADO.
ADO and FOR XML	XML sample. Retrieves result sets into streams
	using the Transact-SQL FOR XML clause.
ADO and Open XML	XML sample. Maps an XML schema to a relational
	schema using annotated schemas.
ADO and XPath	XML sample. Performs an XPath query.
<u>Query</u>	

The ADO samples are not intended to be fully featured applications or demonstrations of the complete range of data access capabilities available through ADO. The samples are designed to cover some basic areas of ADO usage with SQL Server 2000, such as connecting to an instance of SQL Server, querying, editing, and updating a data source, handling multiple result sets, and enumerating provider errors.

To install the samples during SQL Server installation

- 1. On the **Setup Type** page, select **Custom**.
- 2. On the **Select Components** page, under **Components**, select **Code**

Samples.

Samples are installed as a self-extracting file. To extract the samples, doubleclick Unzip_ado.exe, located at C:\Program Files\Microsoft SQL Server\80\Tools\Devtools\Samples\Ado.

Prerequisites

C and C++ samples require Microsoft Visual C++ version 6.0. Visual Basic samples require Microsoft Visual Basic version 6.0.

See Also

Samples

ADO Connection and Error Handling

This sample application demonstrates how to use ADO to connect to and query a database in an instance of Microsoft® SQL Server[™] using the Microsoft OLE DB Provider for SQL Server (SQLOLEDB). Error handling routines are also demonstrated.

Default Location

C:\Program Files\Microsoft SQL Server\80\Tools\Devtools\Samples\ADO\VB\Intro

Running the Sample

Open the Microsoft Visual Basic® project file, Adomain.vbp, and then start the project.

Remarks

Visual Basic version 6.0 includes new data handling tools not covered in this sample, such as the ADO Data Binding Control. Features such as these allow the building of client applications with a minimum amount of code. For more information about building a basic database client using the DataGrid and ADO Data Control, see the MSDN Library at <u>Microsoft Web site</u>

See Also

ADO and Long Data Types (Visual Basic)

This sample application demonstrates how to use ADO and Microsoft® Visual Basic® to display long data types using the **Employees** table in the **Northwind** database. The **Photo** column is an **image** data type, and the **Notes** column is an **ntext** data type.

Default Location

C:\Program Files\Microsoft SQL Server\80\Tools\Devtools\Samples\ADO\VB\Employee

Running the Sample

Open the Visual Basic project file, Employee.vbp, and then start the project.

Remarks

See Also

ADO and Long Data Types (C++)

This sample application demonstrates how to use ADO and Microsoft® Visual C++® to display long data types using the **Employees** table in the **Northwind** database. The **Photo** column is an **image** data type, and the **Notes** column is an **ntext** data type.

Default Location

C:\Program Files\Microsoft SQL Server\80\Tools\Devtools\Samples\ADO\Cpp\Employee

Running the Sample

Open the Microsoft Visual C++® project file, Employee.dsw, and then start the project.

Remarks

See Also

ADO and Long Data Types (Web)

This sample application demonstrates how to use ADO and VBScript to display long data types on an Active Server Page (ASP) using the **Employees** table in the **Northwind** database. The **Photo** column is an **image** data type, and the **Notes** column is an **ntext** data type.

Default Location

C:\Program Files\Microsoft SQL Server\80\Tools\Devtools\Samples\ADO\Web\Employee

Running the Sample

- 1. Register the ActiveX control, FileAccessor.dll, using Regsvr32. For example, if you place the file in C:\Test, register it using this syntax: regsvr32 c:\test\fileaccessor.dll
- 2. In EmployeeSample.asp, set the global variable, TempFileDrive, to an existing directory. This is where the temporary image files are created.
- 3. In EmployeeSample.asp, set the global variable, TempFileHttp, to a URL equivalent to TempFileDrive. For example, if TempFileDrive is set to C:\Inetpub\wwwroot\EmployeeSample\, TempFileHttp might be set to http://Myserver/EmployeeSample/.

Remarks

The Visual Basic code used to create FileAccessor.dll is located at: C:\Program Files\Microsoft SQL

Server\80\Tools\Devtools\Samples\ADO\Web\Employee\Fileaccessor. If you create FileAccessor.dll using these files, it is registered automatically, and Step 1 in Running the Sample is unnecessary. To create FileAccessor.dll using these files:

Open the Visual Basic project file, FileAccessor.vbp.

On the **File** menu, click **Make FileAccessor.dll**.

See Also

ADO Web Application

This sample application demonstrates how to use ADO to build an Active Server Page (ASP) Web application that interacts with an instance of SQL Server. The sample models an inventory management system based on the **Products** and **Categories** tables in the **Northwind** database. The application allows you to view the products and categories, make updates to product information, add new products, and remove products.

Default Location

C:\Program Files\Microsoft SQL Server\80\Tools\Devtools\Samples\ADO\Web

Prerequisites

This ASP sample requires Microsoft® Internet Information Services (IIS) version 4.0 or later, or Microsoft Personal Web Server (PWS). IIS 4.0 for Microsoft Windows NT® Server and PWS for Windows 95, Windows 98, and Windows NT Workstation are included in the Windows NT Option Pack, which can be downloaded from <u>Microsoft Web site</u>. IIS 5.0 is included in Windows 2000 Server, Windows 2000 Advanced Server, and Windows 2000 Datacenter.

Running the Sample

Here are the steps for running the Northwind Inventory Management System application:

- 1. Create a new Web page in IIS or PWS using the directory containing the sample files.
- 2. Open the global.asa file in an editor, add the name of an instance of Microsoft SQL Server[™] to the provider string variable, *ProvStr*, and then save the file.
- 3. Start and browse the Web page from IIS or PWS to view the Northwind Inventory Management System application.

Remarks

To use this sample, you should have a basic understanding of Active Server Pages and IIS. For more information, see the MSDN Library at <u>Microsoft Web</u> <u>site</u>.

See Also

ADO and FOR XML

This sample application demonstrates how to use ADO to build an Active Server Page (ASP) Web application that retrieves result sets into streams using the Transact-SQL FOR XML clause.

Default Location

C:\Program Files\Microsoft SQL Server\80\Tools\Devtools\Samples\ADO\XML\FORXML.asp

Prerequisites

This ASP sample requires Microsoft® Internet Information Services (IIS) version 5.0.

Running the Sample

Add the sample file to a Microsoft Visual InterDev® project, and then click **Start**.

Remarks

See Also

ADO and Open XML

This sample application demonstrates how to use ADO to build an Active Server Page (ASP) Web application that maps an XML schema to a relational schema using annotated schemas.

Default Location

C:\Program Files\Microsoft SQL Server\80\Tools\Devtools\Samples\ADO\XML\OpenXML.asp

Prerequisites

This ASP sample requires Microsoft® Internet Information Services (IIS) version 5.0.

Running the Sample

Add the sample file to a Microsoft Visual InterDev® project, and then click **Start**.

Remarks

See Also

ADO and XPath Query

This sample application demonstrates how to use ADO to build an Active Server Page (ASP) Web application that performs an XPath query.

Default Location

C:\Program Files\Microsoft SQL Server\80\Tools\Devtools\Samples\ADO\XML\Xpath.asp

Prerequisites

This ASP sample requires Microsoft® Internet Information Services (IIS) version 5.0.

Running the Sample

Add the sample file to a Microsoft Visual InterDev® project, and then click **Start**.

Remarks

See Also